

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**COMPUTER MODELING OF CAPTIVE-CARRY MISSILE
SIMULATOR EXPERIMENTS**

by

Wagner A. de Lima Goncalves

September 1998

Thesis Advisor:
Second Reader:

Phillip E. Pace
Robert G. Hutchins

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19981201 123

REPORT DOCUMENTATION PAGE

Form Approved

OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED	
	September 1998	Master's Thesis	
4. TITLE AND SUBTITLE COMPUTER MODELING OF CAPTIVE-CARRY MISSILE SIMULATOR EXPERIMENTS		5. FUNDING NUMBERS	
6. AUTHOR(S) Goncalves, Wagner A. de Lima			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Research Laboratory, Washington, D.C. 20375-5339		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government			
12A. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>The increasing number, diversity and sophistication of anti-ship cruise missiles around the world in the past thirty years have led to sophisticated countermeasures. The Naval Research Laboratory has developed hardware-in-the-loop (HIL) missile simulator technology to assess the effectiveness of electronic attack (EA) countermeasures. These simulators appear in two basic configurations: the closed-loop in an anechoic chamber and the open-loop captive-carry on board a P-3 aircraft.</p> <p>The objective of this thesis was to develop a comprehensive Simulink[®] model representing the two HIL missile simulator configurations. These models were then used to study the influence of each parameter on EA effectiveness, as measured by missile miss distance.</p> <p>The development of this model now makes it possible to compare the seeker responses of the two configurations as well as to have an inexpensive way to test new approaches to combine the closed-loop missile dynamics with the open-loop environment information to obtain more accurate EA effectiveness measurements.</p>			
14. SUBJECT TERMS anti-ship cruise missiles, electronic attack (EA), hardware-in-the-loop, missile, simulations, ASCM Digital Model, EA effectiveness, miss distance		15. NUMBER OF PAGES 313	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL
Standard Form 298 (Rev 2-89) Prescribed by ANSI Std Z39-18 298-102			

Approved for public release; distribution is unlimited.

**COMPUTER MODELING OF CAPTIVE-CARRY MISSILE SIMULATOR
EXPERIMENTS**

Wagner A. de Lima Goncalves
Lieutenant, Brazilian Navy
B.S., Escola Naval (Brazilian Naval Academy), 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author:

Wagner A. de Lima Goncalves
Wagner A. de Lima Goncalves

Approved by:

Phillip E. Pace
Phillip E. Pace, Thesis Advisor

Robert G. Hutchins
Robert G. Hutchins, Second Reader

Dan C. Boger
Dan C. Boger, Chairman

Electronic Warfare Academic Group

ABSTRACT

The increasing number, diversity and sophistication of anti-ship cruise missiles around the world in the past thirty years have led to sophisticated countermeasures. The Naval Research Laboratory has developed hardware-in-the-loop (HIL) missile simulator technology to assess the effectiveness of electronic attack (EA) countermeasures. The simulators appear in two basic configurations: the closed-loop in an anechoic chamber and the open-loop captive-carry on board a P-3 aircraft.

The objective of this thesis was to develop a comprehensive Simulink[©] model representing the two HIL missile simulator configurations. These models were then used to study the influence of each parameter on EA effectiveness, as measured by missile miss distance.

The development of this model now makes it possible to compare the seeker responses of the two configurations as well as to have an inexpensive way to test new approaches to combine the closed-loop missile dynamics with the open-loop environment information to obtain more accurate EA effectiveness measurements.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	CAPTIVE-CARRY VERSUS MISSLE EXPERIMENTS BACKGROUND	1
B.	PRINCIPLE CONTRIBUTIONS	2
C.	THESIS ORGANIZATION.....	3
II.	CLOSED-LOOP MISSILE MODEL	5
A.	ASCM/CAPTIVE-CARRY MODEL.....	5
1.	Tracking Seeker Dynamics	9
a.	<i>Seeker Dynamics Analysis</i>	9
b.	<i>Seeker Description</i>	24
c.	<i>Noise Definitions</i>	24
d.	<i>Noise Generator</i>	31
2.	Autopilot	34
a.	<i>AutoPilot Dynamics Analysis</i>	34
b.	<i>AutoPilot Description</i>	44
3.	Missile/Captive-Carry Dynamics.....	44
a.	<i>Missile Dynamics Analysis & Description</i>	46
b.	<i>Switch Missile/Captive-Carry</i>	57
4.	Time_to_Go (TTG) and Miss_Distance	59
III.	GENERAL TARGET MODEL	63
A.	TARGET GENERATOR.....	63
1.	Original Target Generator.....	70
2.	Ship Generator	70
B.	NULKA GENERATOR	70
1.	Description.....	72
2.	Nulka.m Code	73
3.	Nulka Enabler	75
C.	SWITCH TARGER/NULKA	76
IV.	INITIALIZATION OF THE MODEL PARAMETERS (INITIAL.M CODE)	81
V.	OPEN-LOOP CAPTIVE-CARRY MODEL	89
A.	THE CAPTIVE-CARRY EXPERIMENTS	89
B.	CAPTIVE-CARRY MODEL	89
1.	Captive-Carry Dynamics	89
2.	Selections of the Captive-Carry Experiment	95

ASCM DIGITAL CONTROL.....	97
A. GUIDELINES FOR BUILDING A GUI.....	97
B. GRAPHICAL USER INTERFACES FOR A LARGER NUMBER OF SIMULATIONS.....	98
C. GUI CODES.....	103
1. <i>Update.m</i> Code.....	104
2. <i>Update1.m</i> Code.....	105
3. <i>Store_simulation.m</i> Code.....	105
4. <i>Store_simulation1.m</i> Code.....	105
5. <i>Plot1.m and Plot2.m</i> Code.....	105
D. PROCEDURES TO RUN SIMULATIONS USING <i>MENU</i>	106
VII. COMPARISON OF SEEKER SIGNALS	109
VIII. MISS DISTANCE PREDICTION FROM THE CAPTIVE-CARRY SIMULATION	125
A. INTRODUCTION	125
B. ASCM DIGITAL MODEL CONFIGURATION FOR THE NEURAL NETWORK.....	127
C. TRAINING AND TESTING THE NEURAL NETWORK	127
D. MISSILE TRAJECTORY PREDICTION USING THE NEURAL NETWORK	129
IX. CONCLUSIONS	153
APPENDIX A. ASCM Digital Model.....	155
APPENDIX B. <i>initial.m</i> Code.....	179
APPENDIX C. <i>Noise_Error.m</i>	197
APPENDIX D. <i>nulka.m</i>	201
APPENDIX E. <i>menu.m</i>	205
APPENDIX F. <i>Simulations_plot.m</i>	217
APPENDIX G. <i>update.m</i>	223
APPENDIX H. <i>update1.m</i>	227
APPENDIX I. <i>store-simulation</i>	231
APPENDIX J. <i>store-simulation1</i>	237
APPENDIX K. <i>plot1.m</i>	243
APPENDIX L. <i>plot2.m</i>	247
APPENDIX M. Missile Simulations I	251
APPENDIX N. Missile Simualations II.....	279
APPENDIX O. Neural Network.....	283
LIST OF REFERENCES	293
INITIAL DISTRIBUTION LIST	295

LIST OF FIGURES

2.1.	ASCM Digital Model.....	6
2.2.	Inertial Components in a Missile-Target Engagement	6
2.3.	ASCM/Captive-Carry Digital Model.....	8
2.4.	Seeker Dynamics Block.....	11
2.5.	Block Diagram of the Horizontal LOS Rate.....	12
2.6.	Poles-Zero Diagram of the Horizontal Channel	12
2.7.	Horizontal Channel Root Locus.....	13
2.8.	Missile Trajectory Effects due to <i>ska1</i> Variation.....	16
2.9.	Missile Trajectory Effects due to <i>ska2</i> Variation.....	17
2.10.	Missile Trajectory Effects due to <i>skv1</i> Variation.....	18
2.11.	Missile Trajectory Effects due to <i>skv2</i> Variation.....	19
2.12.	Miss Distance Effects due to <i>ska1</i> Parameter Variation	20
2.13.	Miss Distance Effects due to <i>ska2</i> Parameter Variation	21
2.14.	Miss Distance Effects due to <i>skv1</i> Parameter Variation	22
2.15.	Miss Distance Effects due to <i>skv2</i> Parameter Variation	23
2.16.	Normalized Offset Angle	28
2.17.	Difference Slope versus Sidelobe Level	30
2.18.	Noise Generator	32
2.19.	AutoPilot Model Block.....	37
2.20.	Block Diagram of the Commanded Horizontal Acceleration.....	38
2.21.	Pole-Zero Diagram of the Commanded Horizontal Acceleration.....	38
2.22.	Horizontal Commanded Acceleration Root Locus	39
2.23.	Missile Trajectory Effects due to <i>tal1</i> Variation.....	40
2.24.	Missile Trajectory Effects due to <i>ta2</i> Variation.....	41
2.25.	Missile Trajectory Effects due to <i>tb1</i> Variation.....	42
2.26.	Missile Trajectory Effects due to <i>tb2</i> Variation.....	43
2.27.	Missile/Captive-Carry Dynamics Block	45
2.28.	Missile Dynamics Block	47
2.29.	Velocity Change Sub-Block	49
2.30.	Velocity Change Delay Sub-Block.....	50
2.31.	Missile Trajectory Effects due to <i>tv</i> Variation.....	51
2.32.	Missile Dynamics X-Y Movement	52
2.33.	Missile Trajectory Effects due to <i>tm</i> Variation.....	55
2.34.	Missile Trajectory Effects due to <i>tmv</i> Variation	56
2.35.	Switch Missile/Captive_Carry Dynamics Block	58
2.36.	Time_to_Go (TTG) and Miss_Distance Block.....	62
3.1	General Target Model	64
3.2.	Target Generator Sub-Block	65

3.3	Nulka Generator Sub-Block.....	66
3.4.	Switch Target/Nulka Sub-Block.....	67
3.5.	Target Profiles Available in the ASCM Digital Model: (a) Original Target and (b) Ship.....	69
3.6.	Original Target Generator Model	71
3.7.	Ship Generator Model.....	72
3.8.	Nulka System Components.....	74
3.9.	Nulka Enabler Model	77
3.10.	Switch Target Nulka	79
5.1.	NRL P-3 Captive-Carry Research Aircraft Carrying the HIL Simulators....	90
5.2.	Captive-Carry Dynamics Block.....	92
5.3.	Captive-Carry Trajectory	93
5.4.	Captive-Carry Position Generator Block	94
5.5.	Example of Captive-Carry and Target Trajectories	95
6.1.	GUI Menu	99
6.2.	GUI Simulations_plot	101
7.1.	Seeker Response Comparison for <i>ska1</i> and <i>ska2</i>	111
7.2.	Seeker Response Comparison for <i>ska1</i> and <i>ska2</i>	112
7.3.	Seeker Response Comparison for <i>skv1</i> and <i>skv2</i>	113
7.4.	Seeker Response Comparison for <i>skv1</i> and <i>skv2</i>	114
7.5.	Seeker Response Comparison for <i>enra</i> and <i>enrb</i>	115
7.6.	Seeker Response Comparison for <i>enra</i> and <i>enrb</i>	116
7.7.	Seeker Response Comparison for <i>ta1</i> and <i>ta2</i>	117
7.8.	Seeker Response Comparison for <i>ta1</i> and <i>ta2</i>	118
7.9.	Seeker Response Comparison for <i>rb1</i> and <i>tb2</i>	119
7.10.	Seeker Response Comparison for <i>tb1</i> and <i>tb2</i>	120
7.11.	Seeker Response Comparison for <i>tm</i> and <i>tmv</i>	121
7.12.	Seeker Response Comparison for <i>tm</i> and <i>tmv</i>	122
7.13.	Seeker Response Comparison for <i>tv</i>	123
7.14.	Seeker Response Comparison for <i>tv</i>	124
8.1.	Neural Network Basic Training Process.....	126
8.2.	Neural Network Configuration for Missile Dynamics Prediction	128
8.3.	Log Sigmoid Transfer Function.....	130
8.4.	Linear Transfer Function	130
8.5	Seeker Inputs for the Neural Network (Simulation 5)	134
8.6	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 5...135	
8.7	Seeker Inputs for the Neural Network (Simulation 15)	136
8.8	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 15.137	
8.9	Seeker Inputs for the Neural Network (Simulation 25)	138
8.10	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 25.139	
8.11	Seeker Inputs for the Neural Network (Simulation 35)	140
8.12	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 35.141	

8.13	Seeker Inputs for the Neural Network (Simulation 45)	142
8.14	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 45.	143
8.15	Seeker Inputs for the Neural Network (Simulation 55)	144
8.16	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 55.	145
8.17	Seeker Inputs for the Neural Network (Simulation 65)	146
8.18	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 65.	147
8.19	Seeker Inputs for the Neural Network (Simulation 75)	148
8.20	Missile Trajectory Comparison – Actual x NN Prediction-Simulation 75.	149
8.21	Seeker Inputs for the Neural Network (Simulation 8)	150
8.22	Missile/Captive-Carry Prediction Comparison – Simulation 8	151

LIST OF TABLES

2.1	Estimated Seeker Parameters	27
4.1.	ASCM Digital Model Default Parameters	82
4.2.	ASCM Digital Model Initial Values Range.....	86
8.1.	Miss Distance Comparison between the Actual Threat and the NN Prediction	132
8.2.	Miss Distance Comparison between the Actual Threat and the NN NN Prediction	133

ACKNOWLEDGEMENT

It is with a sincere heart and deep gratitude that I thank Professor Phillip E. Pace for being my thesis advisor. I am thankful for his wise counsel and insight as well as his patience in reviewing several model prototypes and thesis drafts.

Also, I would like to thank Colin Cooper for his help in the design of the graphical user interfaces, Professor Monique Fargues for her assistance with the Neural Network, Professor Robert G. Hutchins for his stimulating suggestions regarding this thesis, and Nancy Sharrock for her patience during the thesis review process.

Nevertheless, helpful discussions with friends such as Maj (BAF) Silvino, LCDR (BN) Cardoso, Maj (BAF) Pontes, CPT (BAF) Breno, LT Marcia Sonon, LTJG (PN) Abrantes, and Ensign (BAF) Taranti also contributed to the enrichment of the subjects covered in this thesis.

Finally, but really most importantly, I thank my lovely wife, Marcia, my daughter, Isabela, and little Gabriel for their support during this challenging phase in my naval career.

I. INTRODUCTION

A. CAPTIVE-CARRY VERSUS MISSILE EXPERIMENTS BACKGROUND¹

Active threat missile systems are one of the warfighter's most serious concerns. Electronic attack (EA) systems play an important role in countering this type of threat. The use of EA, if effective, can alter the flight path of the missile, causing it to miss the target completely. Using hardware-in-the-loop (HIL) missile simulator technology, a cost-effective tool is maintained for testing and evaluating the EA effectiveness in a variety of real-time engagements. The two major HIL simulator test configurations are 1) closed-loop in an anechoic chamber, and 2) open-loop captive-carry on board of an aircraft.

In the closed-loop anechoic chamber configuration, the HIL simulator is mounted on a computer controlled, three axis flight table that simulates the missile flight dynamics by providing roll, pitch and yaw motion. Computer controlled sources located on the anechoic chamber wall (opposite the seeker) simulate a variety of targets, EA and background clutter (synthetic environment). In a real-time simulation, the missile flies toward the target using the guidance signals from the seeker. Since the simulation must finish in a required time frame, the time step is usually chosen as large as possible while maintaining a stable and accurate simulation. The closed-loop configuration in the anechoic chamber is extremely useful for determining the EA effectiveness as a predicted miss distance (MD).

¹ Based on the class notes/homework assignments of EC3700, Fall Quarter 1997 by P. E. Pace.

In the open-loop captive-carry configuration, the simulators are installed on a specially configured aircraft that flies an inbound pattern toward the target/EA system. The captive-carry configuration has the advantage of providing the response of the seeker to an actual terminal phase environment where the natural interference(e.g., background clutter), test geometry, and target scintillation may cause less than predictable behavior. Since the captive-carry simulator does not maneuver, the randome-induced seeker pointing error are different than those experienced by a real missile seeker due to the differences in incident angles of the received signals. Also, multi-path fades will occur at different rates and times due to the large difference in platform velocities.

The captive-carry results contain the seeker angles and the seeker measured range-to-target but contains no missile-dynamics. Consequently, the seeker response does not directly imply an EA effectiveness. For example, important EA techniques such as chaff used in a seduction mode involve time play out issues over several minutes (deployment, bloom, deception). To calculate a predicted MD, the captive-carry seeker response must be modified in combination with a model of the missile dynamics to account for any missile movement that might have occurred over the engagement time- a complicated process. Other high bandwidth EA techniques, however, cause a violent seeker response within a short period of time (e.g., range gate pull-off to cross polarization). For these cases the seeker response may be used directly to calculate a MD.

B. PRINCIPAL CONTRIBUTIONS

The first relevant achievement of this thesis is to provide a Simulink[©] model capable of comparing the closed-loop and open-loop simulations in a compact digital

model. The description of the missile processing (how the missile guides itself to the target) and the mathematical models that are used to evaluate the missile's performance are provided as well as the description of the captive-carry platform and target models.

As a second achievement of this is the design of a MATLAB Graphical User Interface (GUI) to create a friendly and non-time consuming interface between the user and the model parameters in order to run a massive number of simulations. These simulations are required for a comprehensive model analysis, for comparing seeker responses and neural network training, testing and evaluation.

C. THESIS ORGANIZATION

In Chapter II, the part of the ASCM Digital model related with the missile (closed-loop) simulation is described in detail as is the effect of each model parameter on the missile trajectory and miss distance. When necessary, each parameter associated with a missile component is modeled using a block diagram. From that block diagram, the system transfer function is obtained and, from its root locus the theoretical missile performance is predicted.

Chapter III describes the two types of targets that can be simulated with the ASCM Digital Model including the generation of a Nulka decoy. Chapter IV discusses the MATLAB[©] parameter initialization code *intial.m*. Chapter V describes the major differences between the closed-loop and open-loop configurations. Chapter VI describes the Graphical User Interface (GUI) *Menu* and *simulations_plot*. The GUI *Menu* executes several functions such as running the pre-set missile and captive-carry simulations, storing the model results for post-simulation analysis and calling the other GUI,

simulations_plot, for graphing the players (missile, captive-carry and target) of each engagement as well as the seeker performance results (position, rates and accelerations) in each scenario. Chapter VII creates a tactical scenario to run simulations required to compare the seeker responses of the two configurations. In Chapter VIII, a neural network (NN) is created, trained with the closed-loop configuration data and, after that, a missile prediction trajectory is made using the open-loop captive-carry data. Finally, Chapter IX discusses the results obtained in this thesis.

II. CLOSED-LOOP MISSILE MODEL

A. ASCM/CAPTIVE-CARRY MODEL

The purpose of this chapter is to describe the ASCM Digital model (Appendix A) using Simulink[©] 2.1, a MATLAB[©] toolbox. This model was created to simulate a closed-loop engagement by a sea-skimming missile and, depending on the model configurations in the *initial.m* code (Appendix B), also simulates the corresponding open-loop captive-carry experiment. This section will discuss the closed-loop engagement.

The upper level block diagram of the ASCM Digital model is shown in Figure 2.1. It models the tactical engagement represented in Figure 2.2. The inertial components are generated to run the simulation by repositioning the missile in relation to the target position. The inertial parameters are the position differences in the X, Y and Z directions (R_{TM-X} , R_{TM-Y} , R_{TM-Z} , respectively), the magnitude of the distance vector between the target and the missile (R_{TM}), and the horizontal line of sight (L.O.S.) in azimuth (ϕ) and the vertical L.O.S. in elevation (θ).

The formulas below represent the inertial values for the ASCM Digital model:

$$R_{TM-X} = R_{TX} - R_{MX} = \Delta X \quad (1)$$

$$R_{TM-Y} = R_{TY} - R_{MY} = \Delta Y \quad (2)$$

$$R_{TM-Z} = R_{TZ} - R_{MZ} = \Delta Z \quad (3)$$

$$R_{TM-XY} = \sqrt{R_{TM-X}^2 + R_{TM-Y}^2} \quad (4)$$

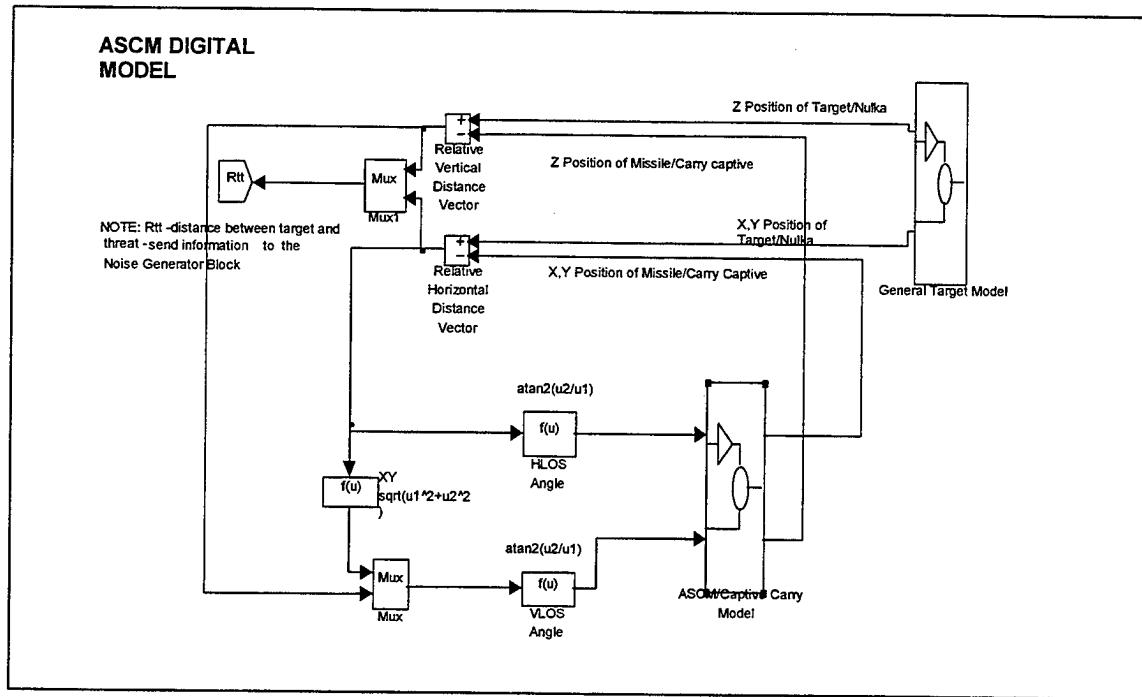


Figure 2.1. ASCM Digital Model

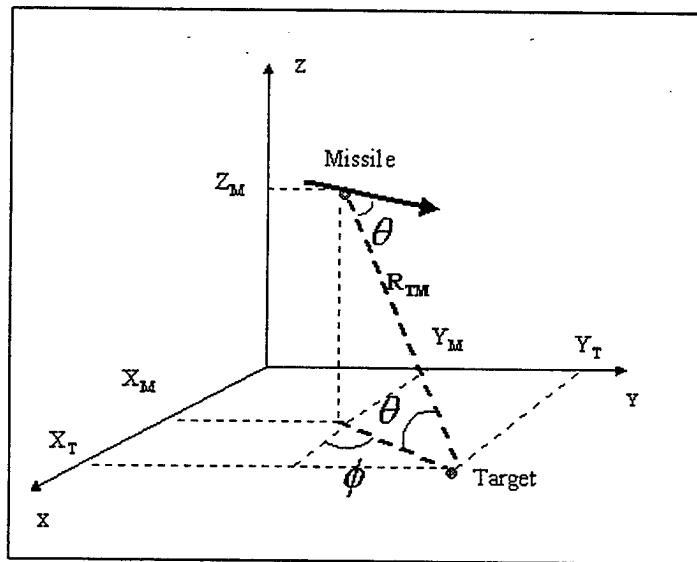


Figure 2.2. Inertial Components in a Missile-Target Engagement

$$\phi = \tan^{-1}(\Delta Y / \Delta X) \quad (5)$$

$$\theta = \tan^{-1}(\Delta Z / R_{TM-XY}) \quad (6)$$

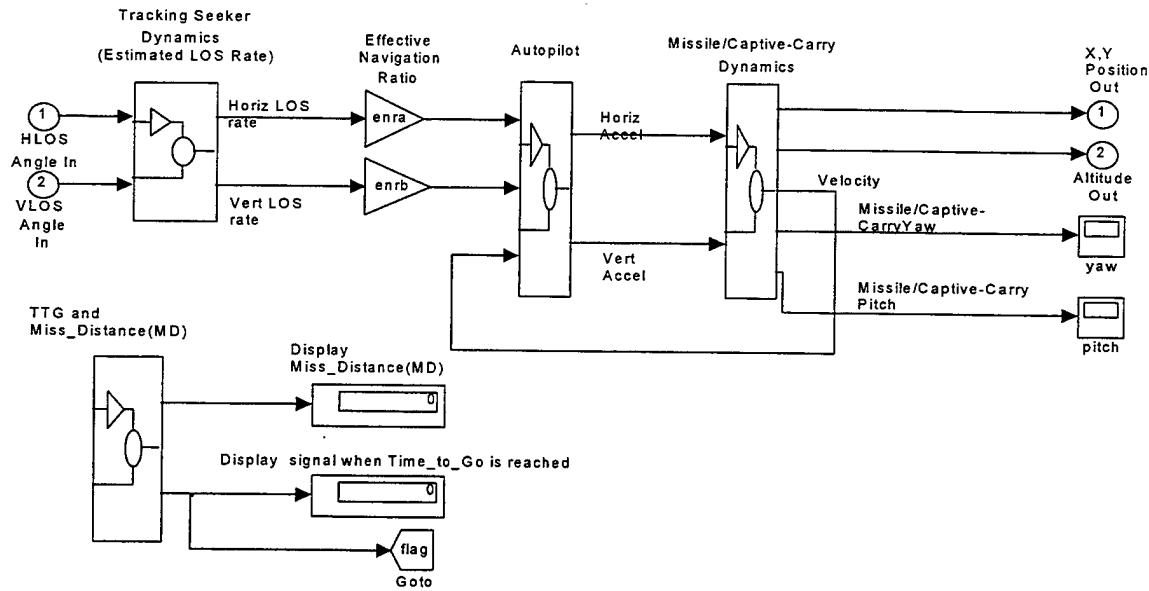
The ASCM digital model receives the missile position from the block ASCM/Captive-Carry model and the target position from the General Target model. The position differences are separated into two channels. One for the X-Y direction (Eqs.1 and 2) and the other for the Z direction (Eq. 3). After the calculation of ΔX , ΔY and ΔZ by the Simulink© summation blocks, these values are combined accordingly in Eqs. 4, 5 and 6 and the final results are the horizontal L.O.S. (ϕ) in and the vertical L.O.S. (θ). These are the inputs of the ASCM/Captive-Carry model.

In the ASCM/Captive-Carry model, estimates of ϕ and θ are introduced in the Seeker Dynamics block (Figure 2.3), where the seeker dish horizontal and vertical L.O.S. rates ($\dot{\phi}$ and $\dot{\theta}$) are obtained. Seeker noise is also added in this block. The ϕ and θ are multiplied by a unitless designer chosen gain known as effective navigation rate in the horizontal (*enra*) and vertical channel (*enrb*).

According to [Ref.1], “the proportional navigation guidance law issues acceleration commands, n_c , perpendicular to the instantaneous missile-target L.O.S., which is proportional to the L.O.S. rate and the closing velocity or

$$n_c = N V_c \lambda \quad (7)$$

ASCM/Captive-Carry Model



SUMMARY:

BLOCK: ASCM / Captive Carry

SUB-BLOCK(s): Tracking Seeker Dynamics
 AutoPilot
 Missile / Captive-Carry Dynamics
 TTG and Miss_Distance

INPUT(s): H.L.O.S. (Horizontal Line of Sight) - ϕ
 V.L.O.S. (Vertical Line of Sight) - θ

OUTPUT(s): Missile X, Y and Z position
 Pitch
 Yaw

AVAILABLE FUNCTION(s): None.

Figure 2.3. ASCM/Captive-Carry Digital Model

where n_c is the acceleration command (in ft/sec²), N' is the effective navigation ratio, V_c is the relative closing velocity (ft/sec) between the missile and the target and λ is the instantaneous L.O.S.. The multiplication ϕ and θ by ***enra*** and ***enrb*** now is understood as a preparation of the L.O.S. rates as inputs to the AutoPilot block (Figure 2.3). In the AutoPilot, the products ϕ ***enra*** and θ ***enrb*** are filtered and multiplied by V_c and sent to the Missile/Captive-Carry Dynamics block where they are used to generate the resulting missile position, velocity magnitude, yaw and pitch. The simulation stops when the missile hits the target or crosses over the target seduced by the Nulka rocket electronic attack (EA).

1. Tracking Seeker Dynamics

a. *Seeker Dynamic Analysis*

The Seeker Dynamics block is shown in Figure 2.4. Before beginning to describe how the L.O.S. rates are calculated in the seeker, a theoretical analysis of the influence of its parameters in the missile trajectory as well as comparing the theoretical results with the results obtained from the model simulations are described. This study was made over the horizontal channel but it is also valid for the vertical channel.

The first step in the analysis was to convert the horizontal channel in Figure 4 into the block diagram on Figure 2.5. From the block diagram

$$e = \lambda_i - \lambda_o \quad (8)$$

$$\dot{\lambda} = \frac{e(ska1) - \lambda(ska2)}{s} \quad (9)$$

$$\lambda_o = \frac{\dot{\lambda}}{s} \quad (10)$$

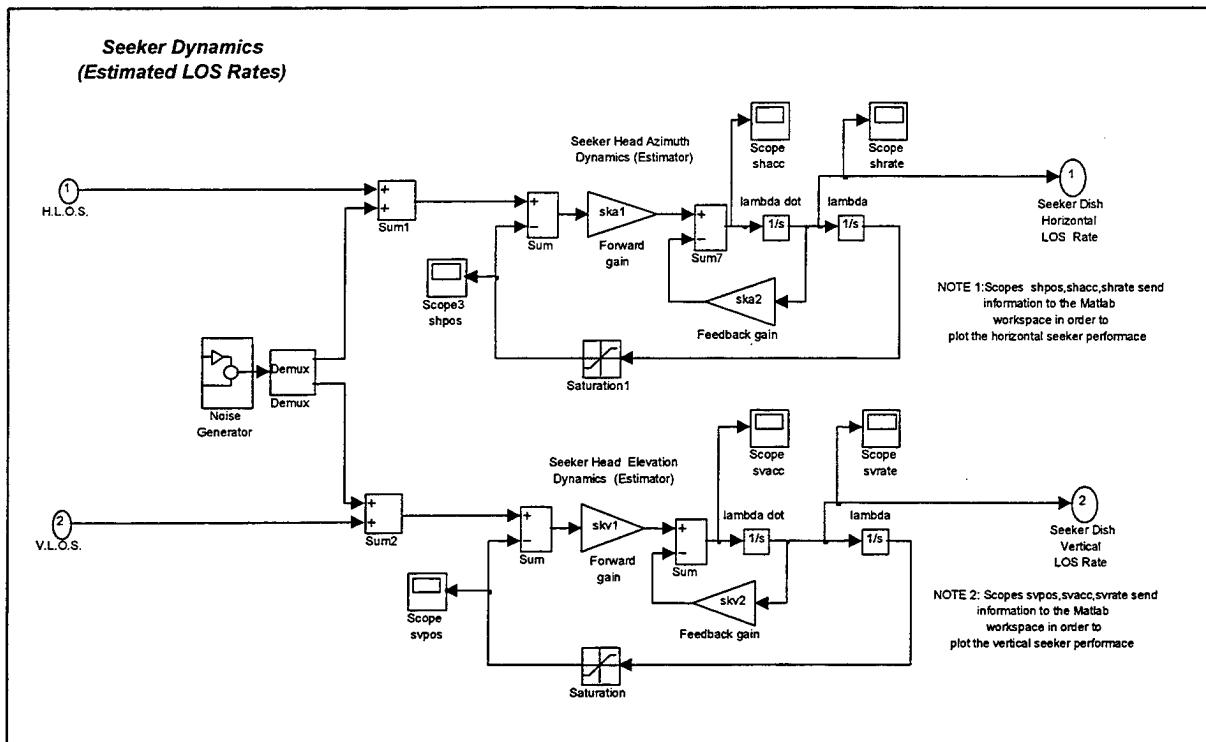
where *ska1* is the seeker forward gain in the Seeker Head Azimuth Dynamics and *ska2* is the seeker feedback gain. After some algebraic manipulations, the channel transfer function is:

$$\dot{\lambda} = \left(\frac{(ska1)s}{(s^2 + (ska2)s + ska1)} \right) \lambda_i \quad (11)$$

and a second order transfer function with one zero, two poles and gain equal to *ska1* is realized. Making the pole-zero diagram shown in Figure 2.6 and using the default values of *ska1*=*ska2*=0.25 a stable system with all poles in the left-hand side of the plane is shown. This generates an oscillation in time due to the imaginary component of the poles (an underdamped system).

To understand the effects of varying the values of the *ska1* necessary to obtain the system open-loop (without feedback) transfer function from Figure 2.5. After some manipulations,

$$\frac{\lambda_o}{\lambda_i - \lambda_o} = \frac{ska\ 1}{s(s + ska\ 2)} \quad (12)$$



SUMMARY:

BLOCK: Seeker Dynamics (Estimated LOS Rates)

SUB-BLOCK (s): None.

INPUT(s): H.L.O.S. (Horizontal Line of Sight) - ϕ
V.L.O.S. (Vertical Line of Sight) - θ

OUTPUT(s): Seeker Dish Horizontal LOS rate - $\dot{\phi}$

Seeker Dish Vertical LOS rate - $\dot{\theta}$

AVAILABLE FUNCTION(s): None.

Figure 2.4. Seeker Dynamics Block

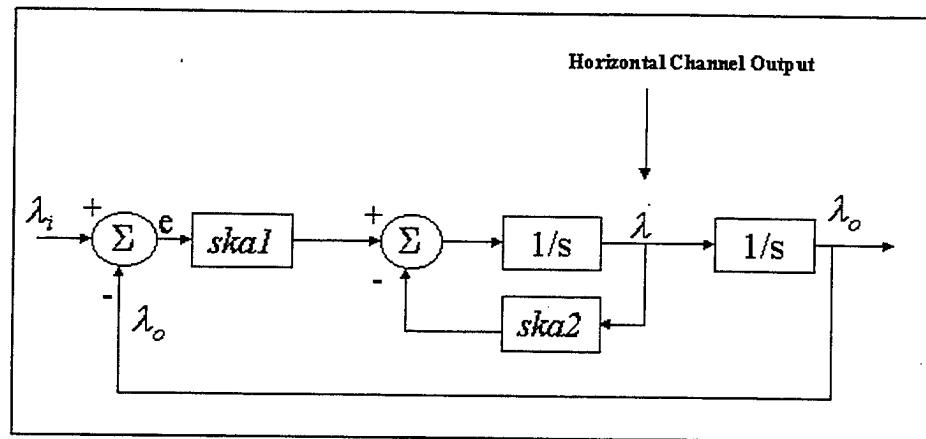


Figure 2.5. Block Diagram of the Horizontal LOS Rate

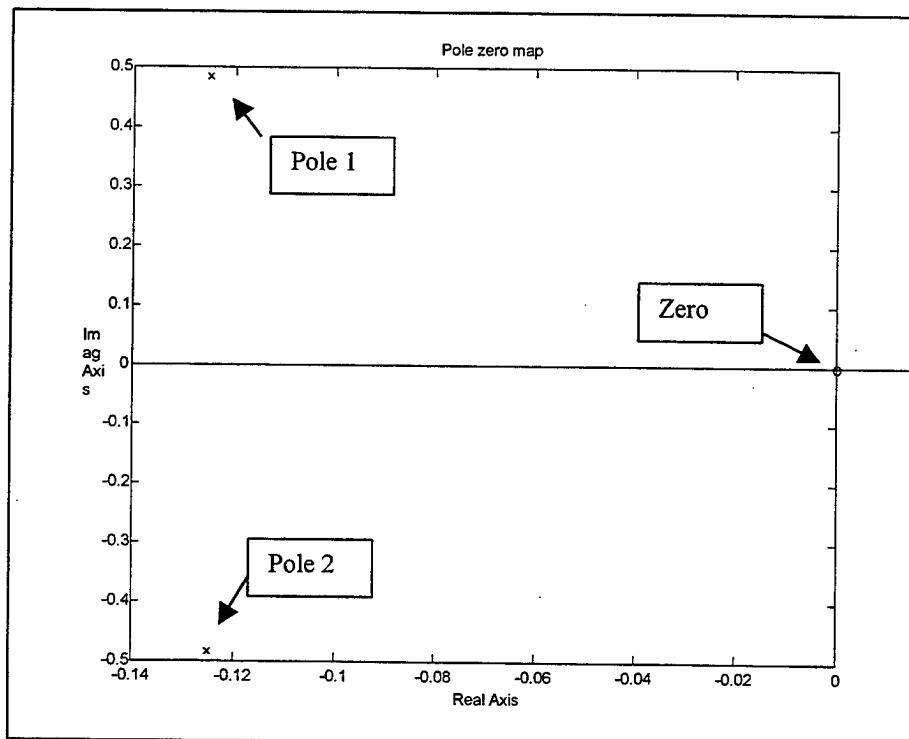


Figure 2.6. Pole-zero Diagram of the Horizontal Channel

where the two poles are located at zero and $-ska2$ and the gain of the system is $ska1$.

The root locus plot is shown in Figure 2.7 for the horizontal channel with $ska1=ska2=0.25$.

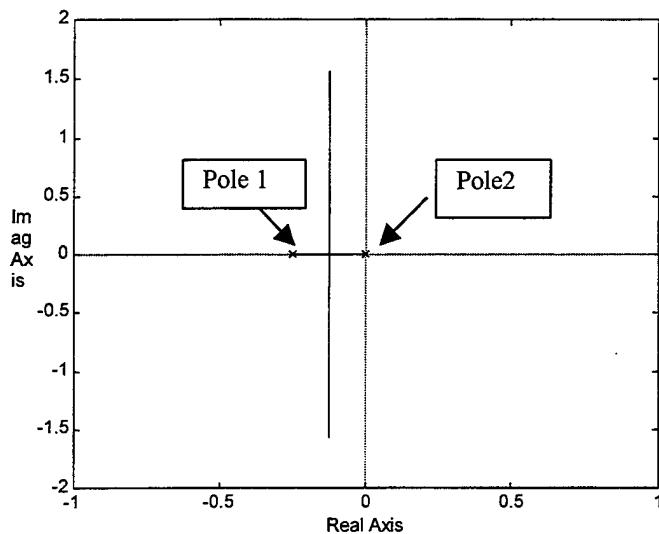


Figure 2.7. Horizontal Channel Root Locus

Analyzing the root locus shown in Figure 2.7, the system output, i.e., the seeker horizontal L.O.S. rate, has a larger natural oscillating frequency (ringing) when the value of $ska1$ is increased. Also, the more the value of $ska1$ is increased the less the system is damped. On the other hand, when the value of $ska1$ is decreased, the seeker L.O.S. rate has few or no oscillations in the stabilization period and the system becomes overdamped.

When the value of *ska2* is increased, one of the system poles becomes further from the vertical axis and, therefore, depending of the value of *ska1*, the system damping is increased. When the *ska2* is decreased, the system poles are closer to the vertical axis and the system becomes less damped. That is, more time is needed for the antenna to stabilize in a position that reduces the error to zero.

From the explanation about the values of *ska1* and *ska2*, it can be concluded that the variation of *ska1* affects the oscillation frequency that occurs and *ska2* affects the damping during that stabilization period. These results can be extended to the *skv1* (seeker forward gain) and *skv2* (seeker feedback gain) parameters for the seeker vertical L.O.S..

The next step is to compare our theoretical results from the root locus study with results obtained from the simulation generated by the ASCM Digital model. In Figure 2.8, the missile trajectory has more oscillations when *ska1*=4 than when *ska1*=0.1, thus confirming our expectations about the performance of the Seeker Head Azimuth Dynamics block in Figure 2.4. In Figure 2.9, it is proven that the larger the *ska2* value is, the less the system response is damped. Thus, the seeker will take longer to stabilize at the correct L.O.S.. All these runs were made using the default values for all the parameters (see Table 4.1), except for the values of *ska1* and *ska2*. These runs also used a COSRO seeker (with seeker noise) with Gaussian shaped antenna beam (see sub-sub-section 1.c). These can be extended for the vertical channel by only changing the name of variables, *skv1* (for forward gain) and *skv2* (for feedback gain). The missile's performance can be observed in Figures 2.10 and 2.11.

Our last step is to observe how the changing seeker parameters affect the miss distance. The miss distance (MD) is the point of closest approach between missile and target. Figure 2.12 shows the MD for the *ska1* variation as a function of the time-to-go (time before the missile hits the target). In the model, time-to-go (*ttg*) is also a variable and, when it is reached, a Nulka decoy, (electronic attack (EA) against sea-skimming missiles) is launched. Note from Figure 2.12 that until *ttg* is equal to 12 seconds, the miss distance is higher for the *ska1* values 0.1 and 4. The missile system with the overdamped seeker, *ska1* = 0.1, responds faster than the underdamped seeker *ska1*=4 when the Nulka is launched. The miss distance is bigger for the overdamped seeker because it more rapidly generates the required rates to reposition the missile towards the Nulka position while the underdamped seeker is still oscillating to reposition the missile to the previous Nulka position. After 12 seconds, the default value *ska1*=0.25 gets a larger miss distance value than the other *ska1* values because it allows the missile to have more maneuvering time to reposition the missile while the target is escaping before the simulation ends. In Figure 2.13, the MD effects of varying *ska2* are presented. In Figures 2.14 and 2.15, the MD effects of varying *sku1* and *sku2* (vertical channel) are observed.

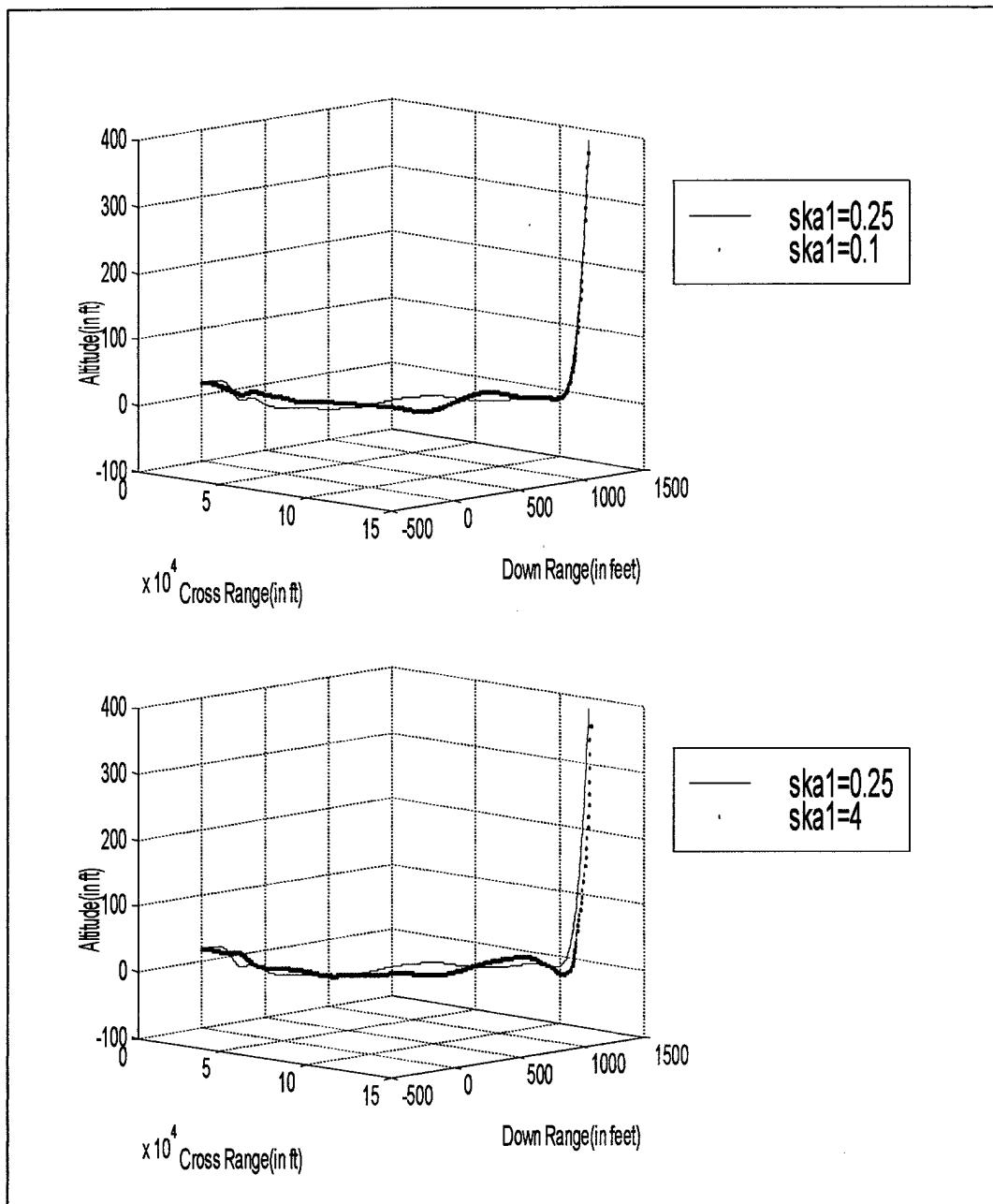


Figure 2.8. Missile Trajectory Effects due to the $ska1$ Variation

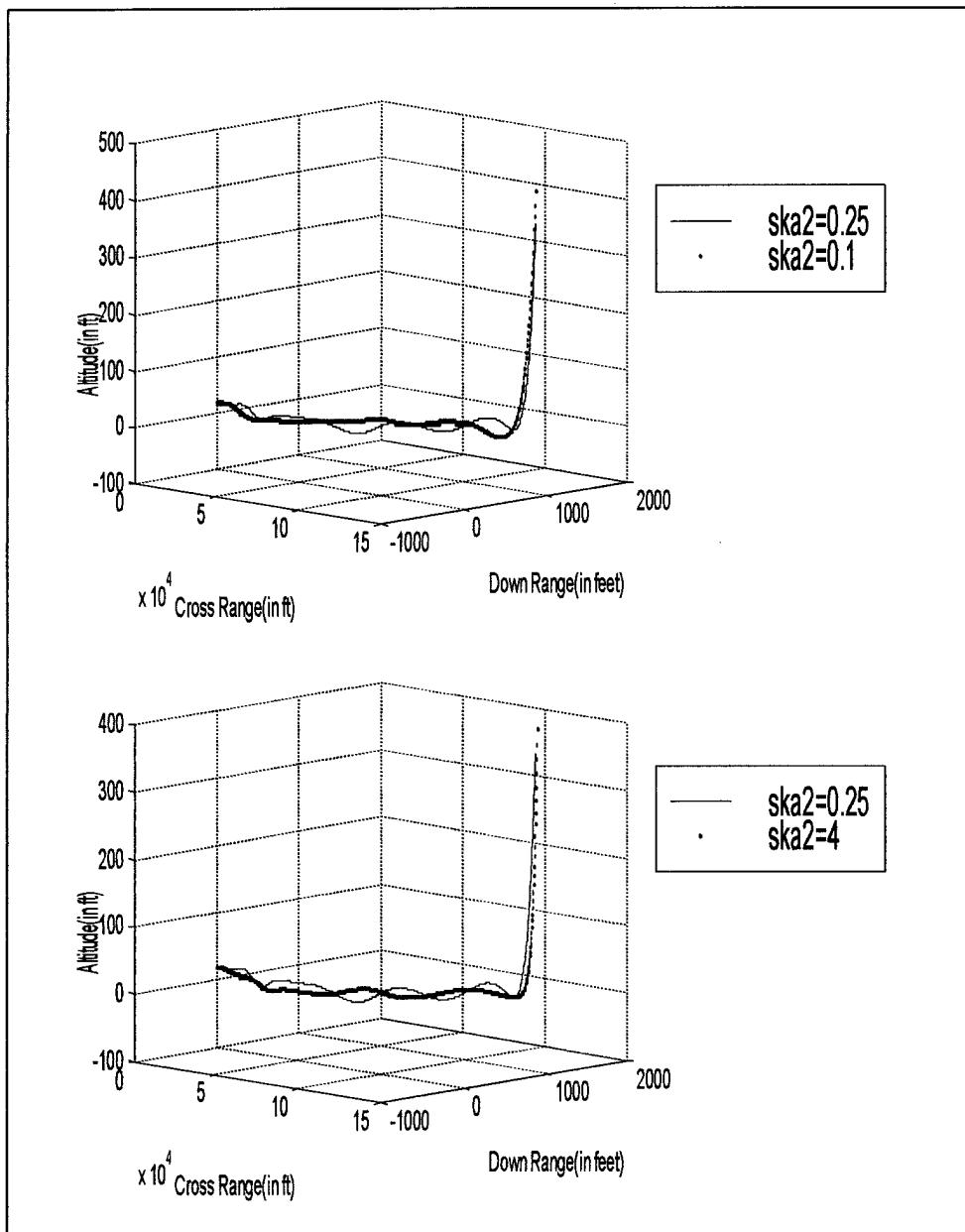


Figure 2.9. Missile Trajectory Effects due to the $ska2$ Variation

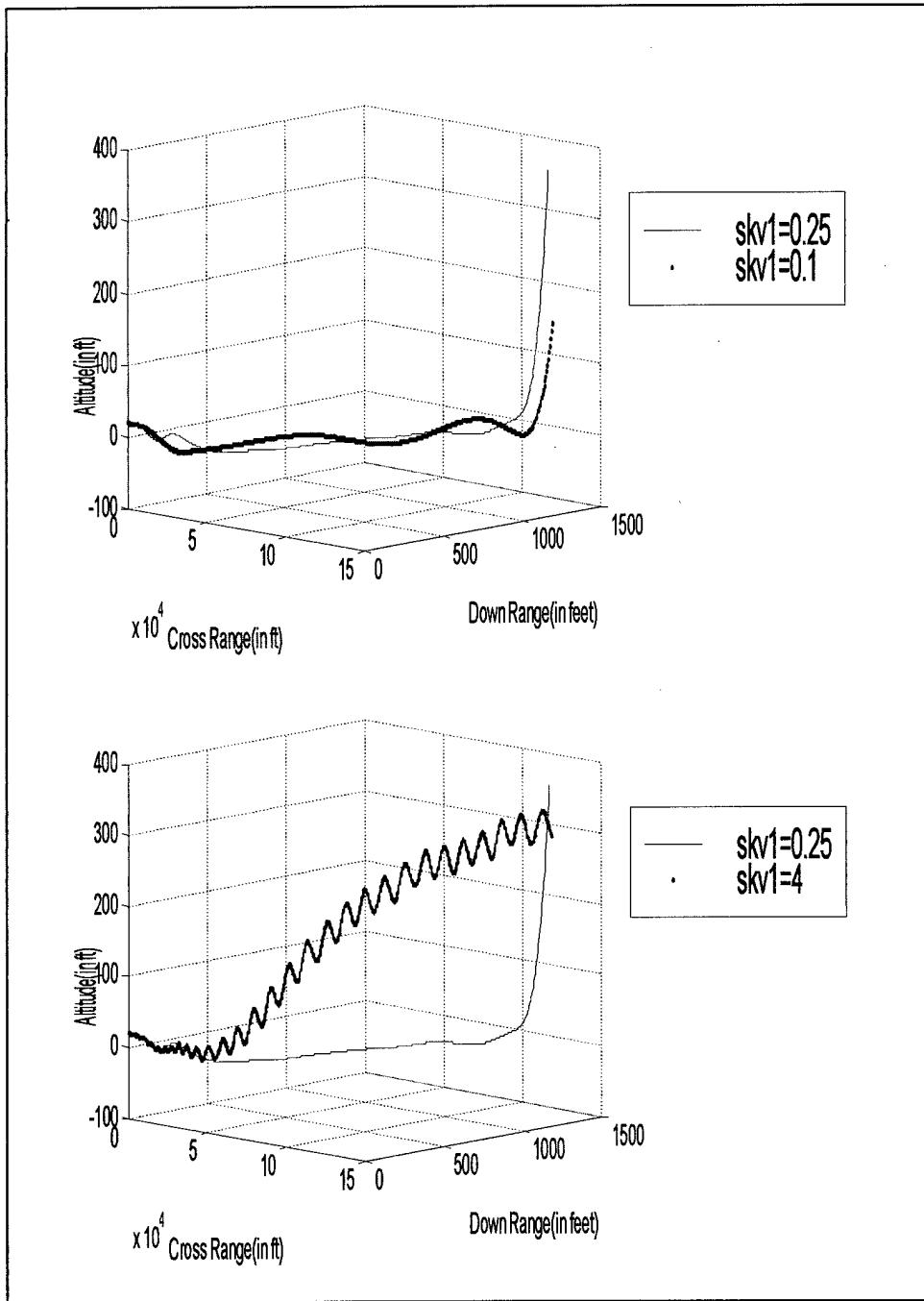


Figure 2.10. Missile Trajectory Effects due to the $skv1$ Variation

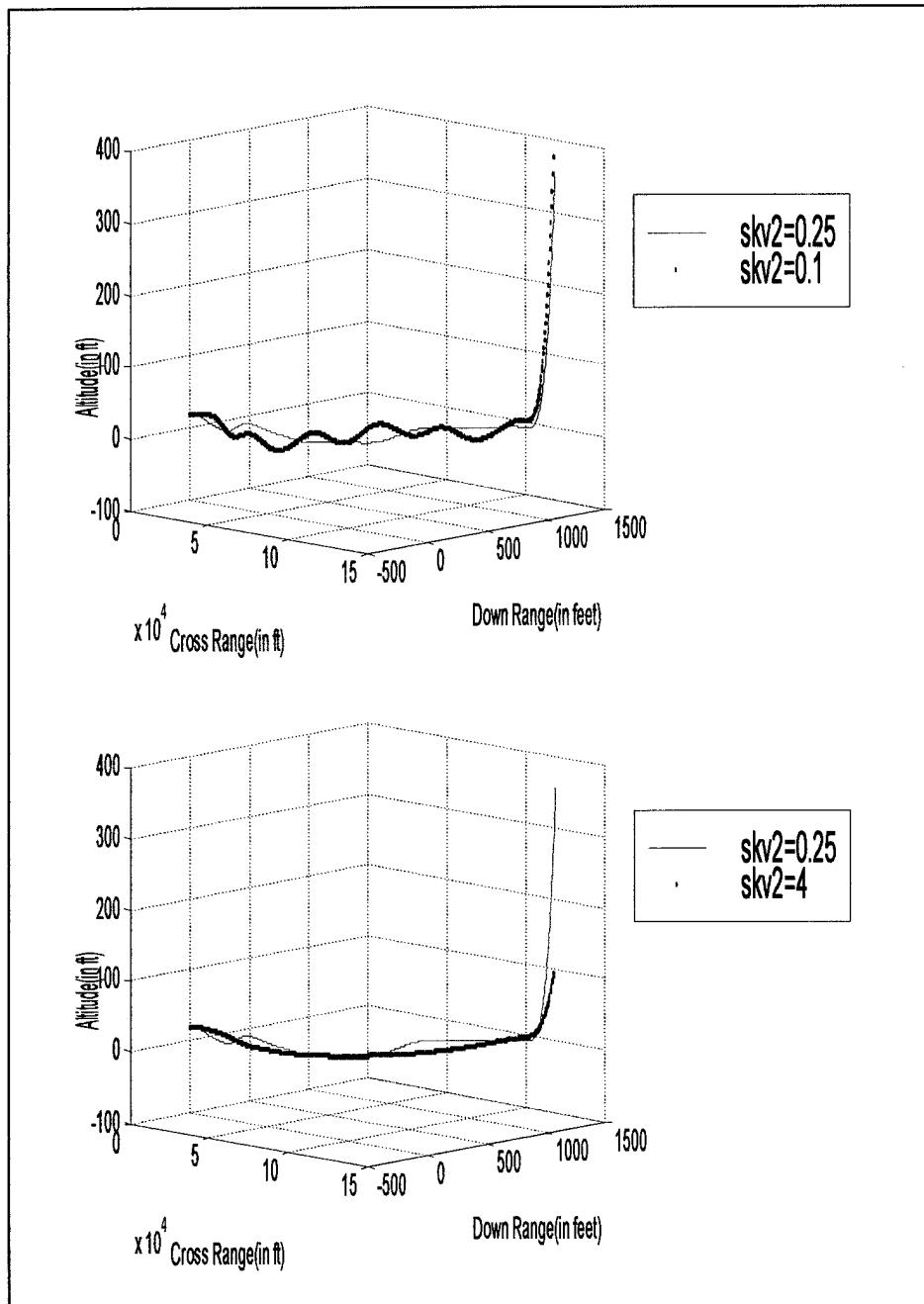


Figure 2.11. Missile Trajectory Effects due to the $skv2$ Variation

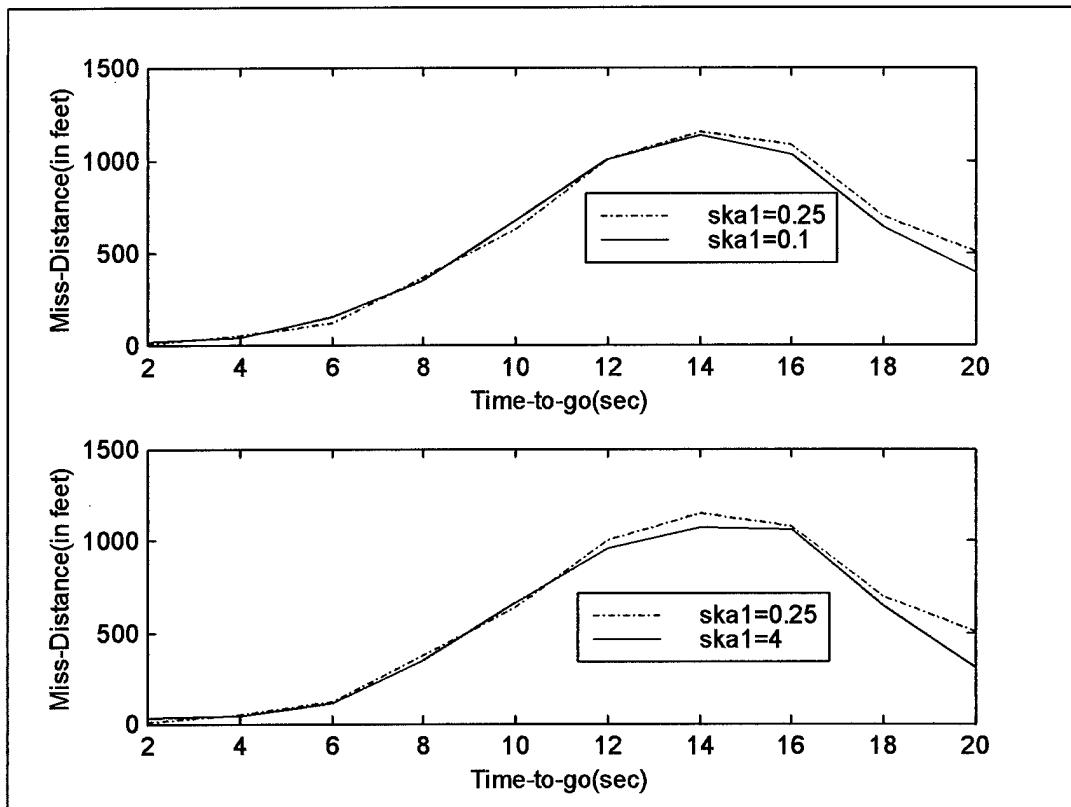


Figure 2.12. Miss Distance Effects Due to $ska1$ Parameter Variation

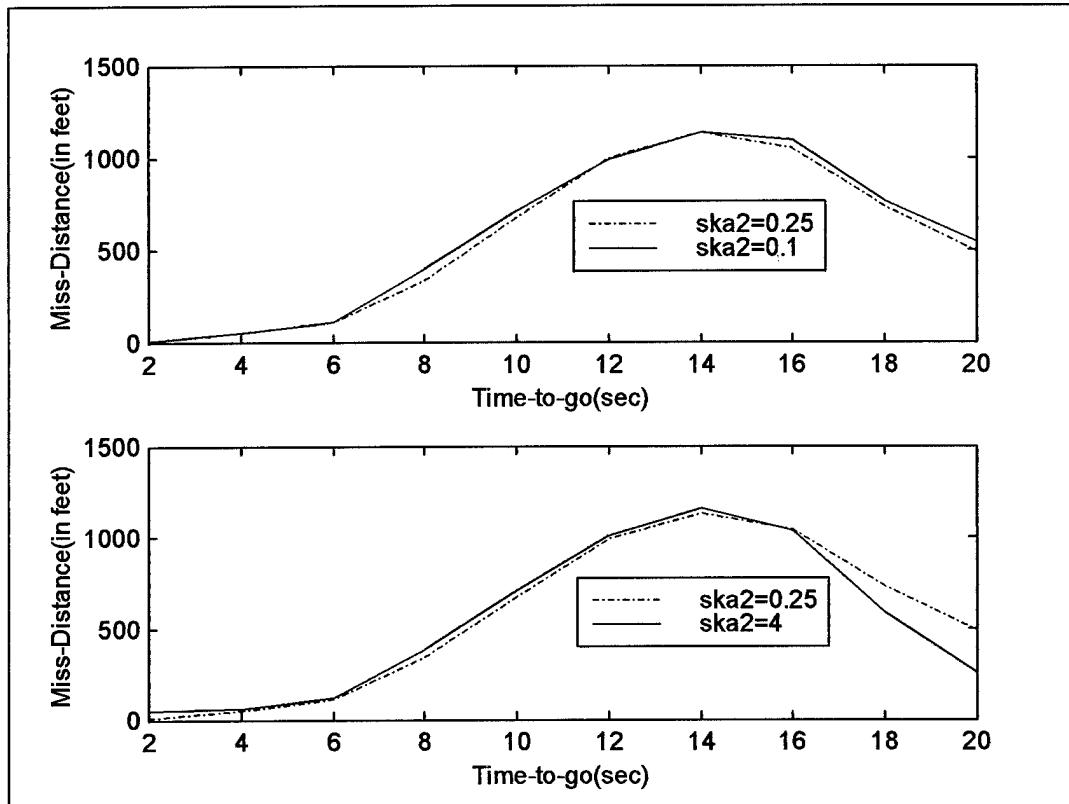


Figure 2.13. Miss Distance Effects due to $ska2$ Parameter Variation

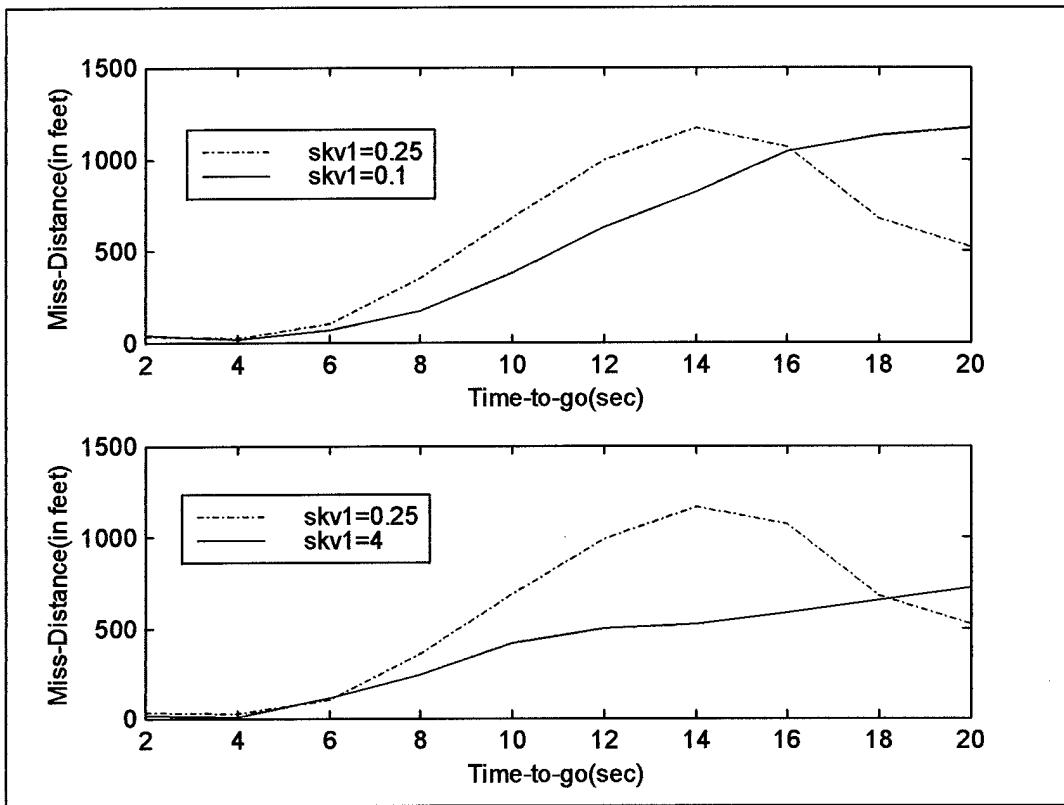


Figure 2.14. Miss Distance Effects due to $skv1$ Parameter Variation

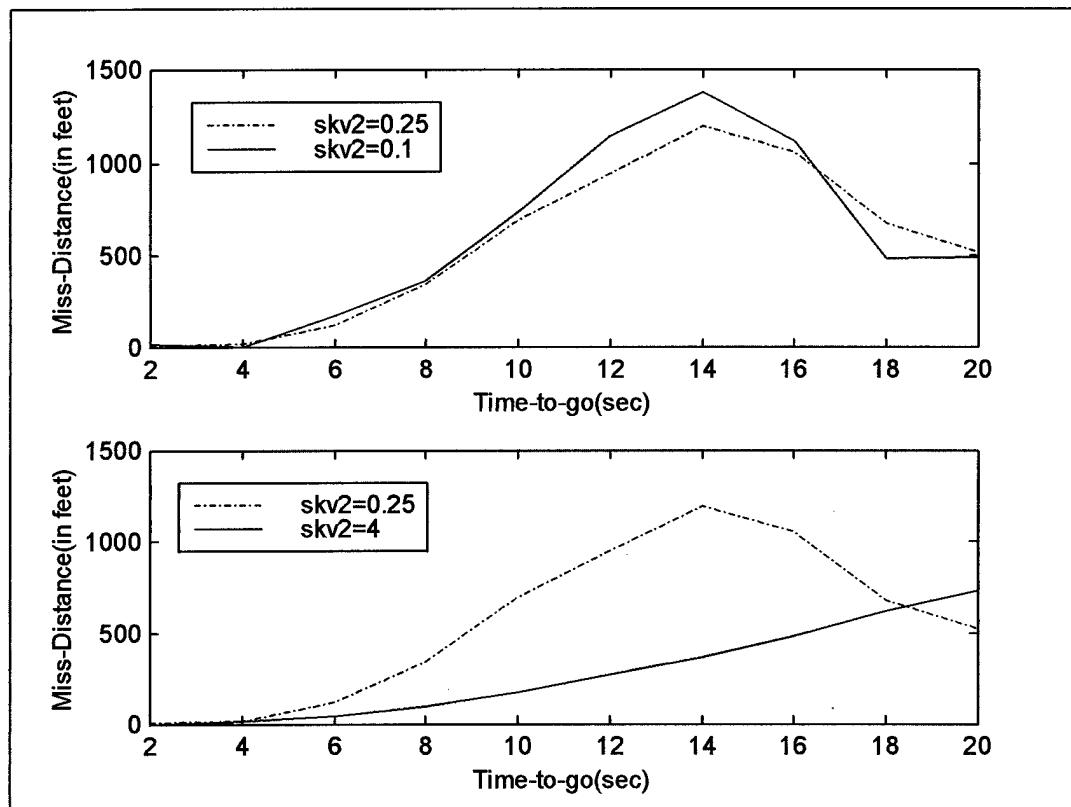


Figure 2.15. Miss Distance Effects due to $skv2$ Parameter Variation

b. Seeker Description

The Seeker Dynamics block shown in Figure 2.4 receives its inputs, H.L.O.S.(ϕ) and V.L.O.S.(θ) from the ASCM Digital model. These inputs have a random noise added by a Noise Generator block. After the addition of noise, the H.L.O.S. and V.L.O.S. are processed by the Seeker Head in order to calculate the new seeker dish position. When the seeker dish reaches its final position, the errors in both horizontal and vertical channels are minimized. The Seeker Head Azimuth and Elevation Channels are second order linear systems.

The calculated position of the antenna is fed back through a field-of-view saturation block whose limit values are set by the variables *sekhlim* ($\pm 15^\circ$) for the horizontal channel and *sekvlim* ($\pm 15^\circ$) for the vertical channel. The system outputs to the AutoPilot are the seeker horizontal L.O.S. rate ($\dot{\phi}$) and the seeker vertical L.O.S. rate ($\dot{\theta}$).

c. Noise Definitions

The noise concepts for this model as well as how the noise value is calculated from the selected seeker parameters for the horizontal and vertical channels, are discussed here and in the next sub-sub-section.

In the missile and captive-carry model, two basic types of seekers are used. The first is the conical scan on receive only seeker (COSRO¹) and the second is a

¹ A COSRO seeker [Ref. 2] is defined as “An antenna...with a radiant element which is caused to move in a small circular orbit about the focus of the antenna with or without

monopulse seeker². In Table 2.1 the estimated seeker parameters for the two seeker types are introduced [Ref. 4]. Table 2.1 also presents the seeker model variables and their default values for our noise calculations. The signal-to-noise ratio for a single pulse $(S/N)_m$ as a function of the distance between the seeker and the target is

$$(S/N)_m = \frac{P_p G \sigma e^{-2\alpha R} \lambda^2}{(4\pi)^3 R^4 (kTBF)} \quad (13)$$

where,

P_p = peak power

G = antenna gain, same gain for transmission and reception

σ = target radar cross section

α = attenuation coefficient

R = distance between missile and target

λ = wavelength

k = Boltzman's constant = 1.380658×10^{-23} Joules K⁻¹

T = receiver temperature in Kelvin

KT = 4.14×10^{-21} Watts / Hz at 300 K

B = receiver noise bandwidth

change of polarization. The radiation pattern is in the form of a beam that traces out a cone centered on the reflector axis.”

² A monopulse seeker [Ref. 3] is defined as “(...)A type of tracking radar that permits the extracting of tracking error information from each received pulse and offers a reduction in tracking errors as compared to a conical-scan system of similar power and size.”

F = noise figure³

With Eq.(13) being presented, the factors that affect the precision of the angular measurement for a COSRO and monopulse seeker can be evaluated. For the COSRO seeker, two major factors affect the seeker angular precision: the error caused by thermal noise over n integrated pulses by the servo loop and the scintillation error. The error caused by thermal noise over n integrated pulses by the servo loop in each coordinate (azimuth and elevation) is given by the formula below from [Ref. 6]

$$\sigma_\theta = \frac{\theta_{3dB} \sqrt{L_k}}{k_s \sqrt{(S/N)_m n}} \quad (14)$$

where,

L_k = cross over loss due to the beam squint⁴

k_s = conical scan error slope

$(S/N)_m$ = single pulse signal-to-noise ratio

n = number of pulses integrated

θ_{3dB} = half power beam width

³ In [Ref. 5], noise figure is defined as "A figure of merit...given by the ratio of the signal-to-noise ratio at the input, S_i/N_i divided by the signal-to-noise at the output, S_o/N_o .

⁴ Crossover loss due to beam squint is a reduction of the gain due to the target being tracked from the beam peak.

Estimated Parameter	COSRO	Monopulse	Variable Names
			c – COSRO m - monopulse
Peak Power, P_p (kW)	250	30	ppc / ppm
Frequency, f (GHz)	9	17	$freqc / freqm$
Antenna gain, G (dB)	33	23.4	$antgainsc / antgainm$
Half Power Beam Width, θ_{3dB} (°)	4.5	8	HPc / Hpm
Noise Bandwidth, B (MHz)	10	10	$noibwc / noibwm$
Noise Figure, F (dB)	11	9.5	$noifigc / noifigm$
Range resolution (meters)	100	100	rrc / rrm
Number of integrated pulses, n	100	100	$numpulc / numpulg$
First side lobe ratio (dB)	- x -	24	$fslrm$
Target radar cross section, σ (m^2)	3000	3000	$crossc / crossm$
Attenuation Coefficient, α (dB/km)	0.055	0.055	$alphac / alpham$

Table 2.1. Estimated Seeker Parameters (After [Ref. 5])

Figure 16 shows k_s and L_k as a function of the normalized squint (offset) angle, θ_k / θ_{3dB} .

The scintillation error, i.e., the reduction of the gain because the target is tracked off the beam peak, is given by [Ref. 6]

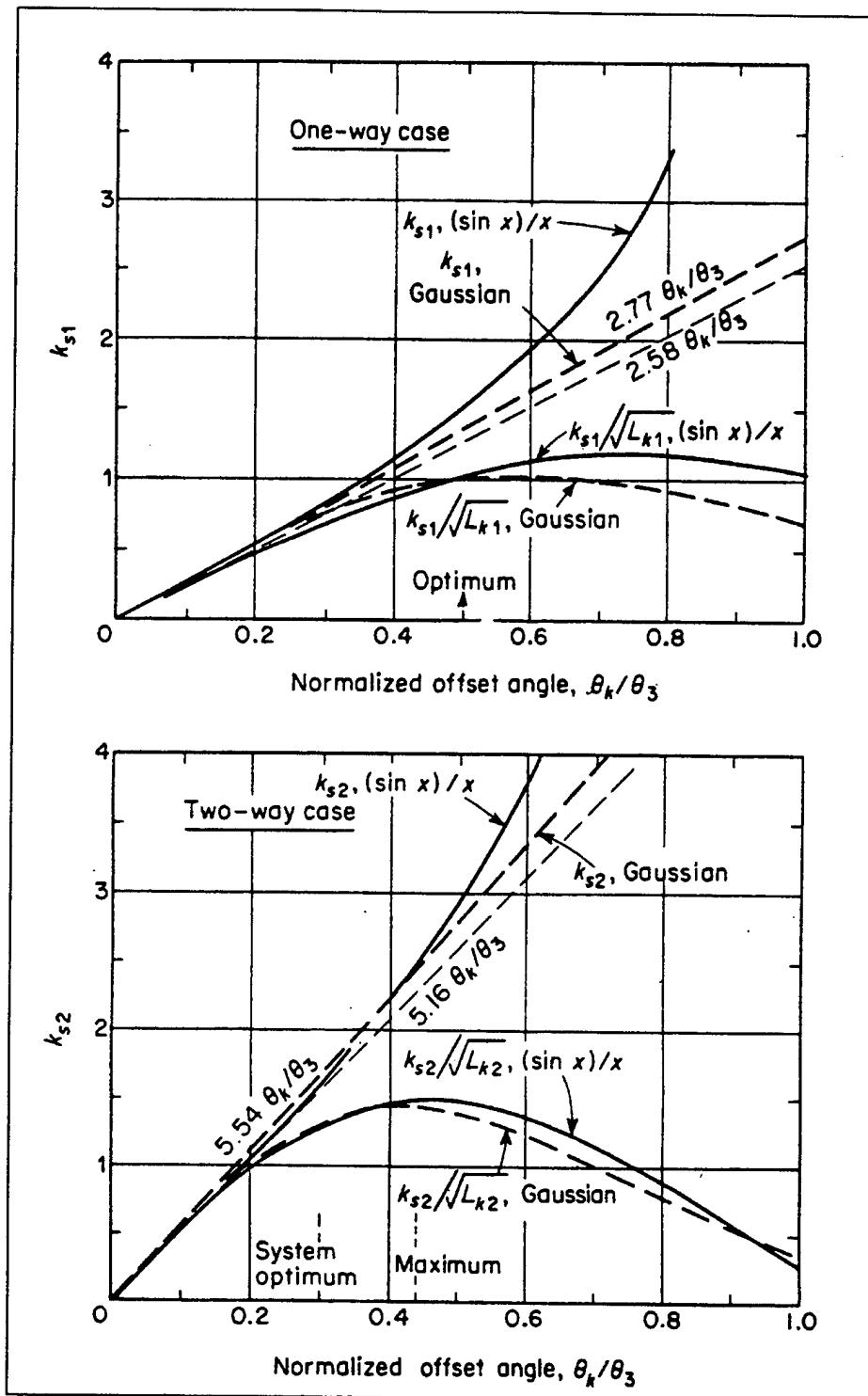


Figure 2.16. Normalized Offset Angle [Ref. 6, p. 389]

$$\sigma_s = 0.225 \left(\frac{\theta_{3dB}}{f_s k_s} \right) \sqrt{\frac{\beta_n}{t_c}} \quad (15)$$

where,

β_n = servo bandwidth

f_s = nutation frequency

t_c = envelope correlation time

The total error in the precision of the angular measurement for a conical scan seeker is found by adding Eqs. (14) and (15). The error in the angular measurements, $\delta\theta$ (vertical) and $\delta\phi$ (azimuth), is assumed to be Gaussian distributed and given by

$$\delta\theta = \delta\phi = \frac{1}{\sqrt{2\pi(\sigma_s^2 + \sigma_\theta^2)}} e^{-\frac{1}{2} \left(\frac{\theta - \theta_{TRUE}}{\sigma_s^2 + \sigma_\theta^2} \right)^2} \quad (16)$$

where θ_{TRUE} is the current L.O.S. within the simulation, and σ_θ is a function of the $(S/N)_m$.

For the monopulse seeker, the precision of the angular measurement in a thermal noise environment is given by

$$\sigma_{\theta MP} = \frac{\theta_{3dB}}{k_m \sqrt{2(S/N)_m n}} \quad (17)$$

where k_m is the difference slope and n is the number of pulses being integrated. The value of k_m can be found from the graph shown in Figure 2.17 as a function of the first

side lobe ratio. For a first side lobe ratio equal to 24 dB (Table 2.1) the value of k_m is approximately 1.9. The beam shape of the antenna is assumed to be Gaussian.

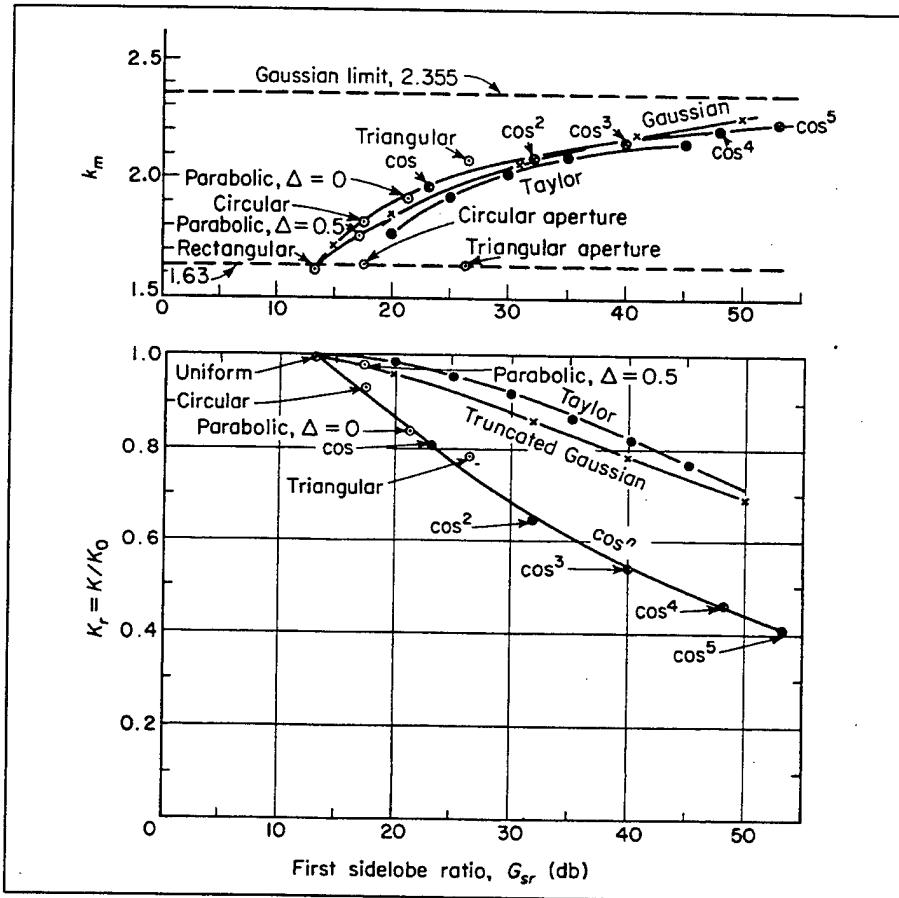


Figure 2.17. Difference Slope Versus Sidelobe Level [Ref. 6, p. 403]

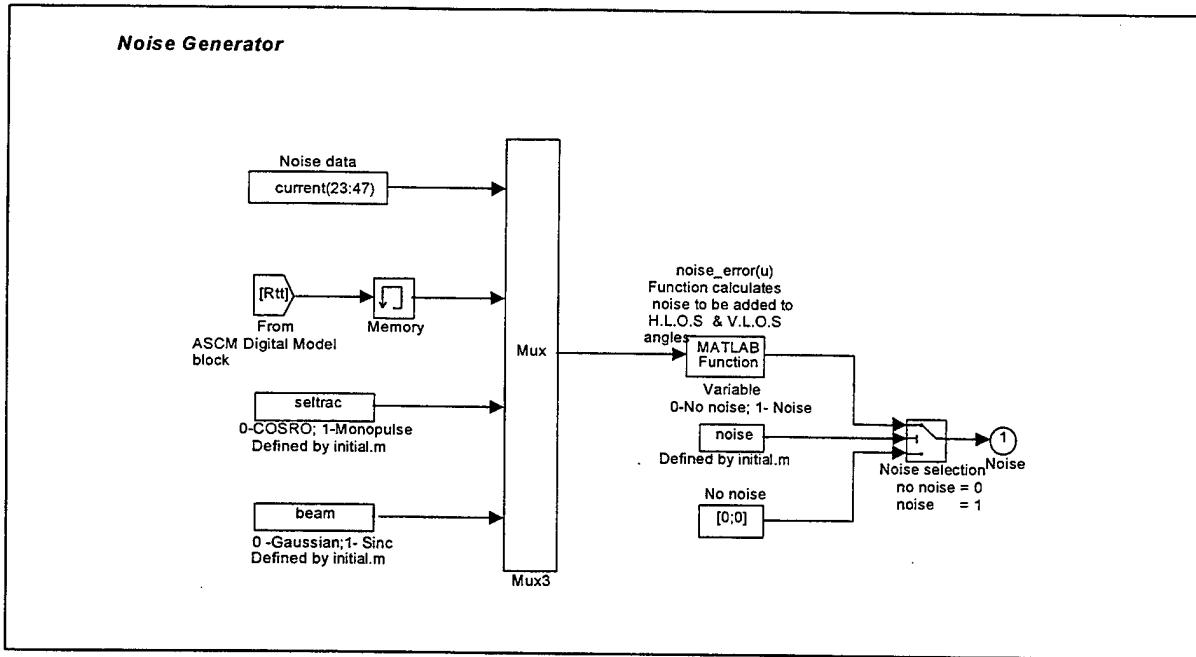
The error in the angular measurement, δ_θ or δ_ϕ , is assumed to be Gaussian distributed and given by

$$\delta\theta = \delta\phi = \frac{1}{\sqrt{2\pi(\sigma_{\theta_{MP}})^2}} e^{-\frac{1}{2} \left(\frac{\theta - \theta_{TRUE}}{\sigma_\theta^2 + \sigma_\theta^2} \right)^2} \quad (18)$$

where θ_{TRUE} is the current L.O.S. within the simulation and $\sigma_{\theta_{MP}}$ is a function of the $(S/N)_m$. Eqs. 13, 14, 15, 16, 17 and 18 as well as the values in Table 2.1 are used for the noise calculation in the Noise Generator block (*noise_error.m*, Appendix C).

d. Noise Generator

The Noise Generator block in Figure 2.18 is the Seeker Dynamics sub-block responsible for generating the noise to be added to the horizontal and vertical L.O.S.. It receives as inputs the vector *current*, the distance between the threat (missile or captive-carry) and the selected target or the Nulka decoy (function of the *ttg*), and the general switches *seltrac* and *beam*. The vector *current* introduces the 25 values related to the variables established in Table 2.1 for the COSRO and monopulse seekers. Those values are stored in the initialization structure *I* and their values are assigned to the vector *current* as a function of the simulation number selected in the GUI (Graphical User Interface) *Menu*. The variable *Rtt* sends the position differences in X, Y and Z directions between the threat and the target from the ASCM Digital model.



SUMMARY:

BLOCK: Noise Generator

SUB-BLOCK(s): None.

INPUT(s): Vector current. It introduces the following parameters:

For a COSRO Seeker - *ppc, freqc, antgains, HPc, noibwc, rrc, numpulc, nosangc, envc, nutfreqc, serbw, crossc, alphac*;

For a monopulse Seeker - *ppm, freqm, antgainm, HPm, noibwm, rrm, fslrm, crossm, alpham*;

Rtt - distance between the threat and the selected target

seltrac - general switch that allows selection between the COSRO or monopulse seeker,

beam – general switch that allows selection between Gaussian or sinc beam shape for the COSRO seeker,

noise – general switch that enables or not the noise calculation results out of the Noise Generator block.

OUTPUT(s): Vector 2x1 [$\delta\phi; \delta\theta$] introduces random noise to the horizontal and vertical L.O.S in the block Seeker Dynamics.

AVAILABLE FUNCTION(s): *noise_error.m* (See Appendix C)

Figure 2.18. Noise Generator Model Block

The general switch ***seltrac*** allows selecting between the type of seeker available in the model (COSRO seeker or monopulse seeker). The default seeker was chosen to be the COSRO seeker. When the COSRO seeker is selected, the value of ***seltrac*** is set to zero and when the monopulse seeker is selected its value goes to one. The general switch ***beam*** allows selection between the Gaussian (default, ***beam*** = 0) or the sinc function beam shape (***beam*** = 1).

The above values are combined in one matrix (30,1) by a Simulink[©] multiplexer and the resultant vector is introduced in the function *noise_error.m* shown in Appendix C. Depending on the value of ***noise***, the results from the Noise Generator block are sent (***noise*** = 1) or not (***noise*** = 0) to the Seeker Dynamics block to be added to the horizontal and vertical L.O.S..

The *noise_error.m* calculates the distance between the threat and the target, which is the range R in Eq. (13). As a second step, it assigns the parameter values for the COSRO and monopulse seeker to auxiliary variables for further calculations of the error in the precision of the angular measurements. The next step is to calculate the signal (power) received by the seeker with the parameters of the chosen seeker (defined by the value of ***seltrac***). After that, the noise is calculated and finally the $(S / N)_m$ for a single pulse is calculated.

If the selected seeker is COSRO, a MATLAB[©] function (*polyfit*) is used to extract the values of k_s shown in Figure 2.16 in order to calculate Eqs. 14 and 15. The values of k_s depend on which seeker is used (defined by the variable ***beam***). For the

monopulse seeker, the same MATLAB[©] function is used to get the value of k_m used in Eq. 17 from the curves presented in Figure 2.17.

Eqs. 14 and 15 are used to get the standard deviations due to both thermal and scintillation errors for the COSRO seeker. For the monopulse seeker, Eq. 17 is used to calculate the standard deviation due to the thermal noise. The errors of the angular measurement have a Gaussian distribution. The mean is zero and the standard deviations given by Eqs. 14 and 15 or 17. i.e.,

$$\text{error} = \text{Normal}(\theta_{\text{TRUE}}, \sigma_\theta). \quad (19)$$

Therefore, horizontal and vertical errors from a normal distribution are generated as shown above for each measured angle. These errors, after calculated, are added to the values of ϕ and θ in the Seeker Dynamics block, causing the seeker to behave statistically as a real system.

2. Autopilot

The Autopilot is the block responsible for filtering the horizontal and vertical accelerations that are used in the missile dynamics to calculate the new missile position. Before describing the Autopilot block presented in Figure 2.19, an analysis of how the model parameters in this block affect the missile performance is required.

a. Autopilot Dynamics Analysis

For the Autopilot dynamic analysis, this study will concentrate on the horizontal channel, knowing that the results can be extended to the vertical channel as well. The model variables in this block are **ta1** forward gain and **ta2** feedback gain for

the Commanded Horizontal Acceleration. For Commanded Vertical Acceleration the variables are *tb1* forward gain and *tb2* feedback gain for the vertical channel.

A block diagram of the horizontal channel is shown in Figure 2.20 from which the closed loop transfer function is obtained:

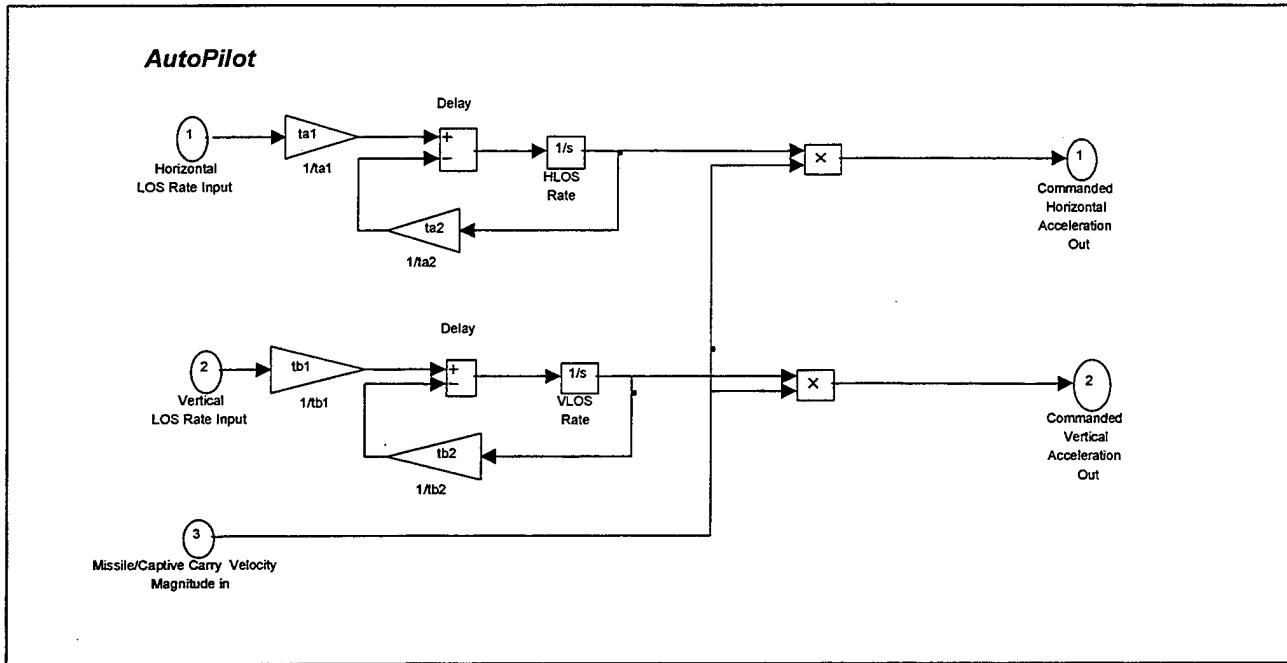
$$\frac{\lambda_i}{\lambda_o} = \frac{1/t_{a1}}{s + (1/t_{a2})} \quad (20)$$

This is a first order system with one pole at $(-1/t_{a2})$ and a gain of $(1/t_{a1})$. Using the default values $t_{a1}=t_{a2}=0.5$, the single pole is at -2 and the gain equals 2. The graph shown in Figure 2.21 shows the pole-zero diagram for the default values. The next step is to see what happens when the values of *ta1* and *ta2* are varied. Making the root locus shown in Figure 2.22 from the open-loop transfer function, when the system gain value goes to infinity, the single pole goes toward the imaginary axis, but, on the other hand, when its value goes to zero, the system pole goes to $-\infty$.

The root locus analysis and Figure 2.20 shows that the closed loop system output (λ_o) is delayed from the system input (λ_i) by an amount $(1/t_{a2})$. The larger the *ta2* value the smaller the delay of the output response to the system input. The smaller the *ta2* value the larger the delay until the system output reaches the value of the system input. The *ta1* value can be understood as a system gain. The larger the *ta1* value the smaller the system amplification and the smaller the *ta1* value, the higher the system amplification. Thus the *ta2* value affects the time in which the commanded horizontal

acceleration will be sent to the missile dynamics. If it takes a long time (*ta2* small), the necessary acceleration to generate the missile position in the Missile/Captive-Carry Dynamics block may not be enough to reposition the missile in the correct path. On the other hand, a large value of *ta2* makes the missile oscillate when trying to track each slight variation in the horizontal L.O.S. rate. A large *ta1* value has a tendency to reduce the amplitude of the horizontal L.O.S. rate, generating, as expected, a commanded horizontal acceleration output in a amplitude lower than the one requested to reposition correctly the missile toward the target, and thus undercorrecting it. For small value of *ta1* the inputs to the missile dynamics are larger than necessary and thus causing an overcorrection in the missile position. The above results are easily extended to the vertical channel as well.

Finally, the analysis above can be verified in the missile trajectories when *ta1* and *ta2* are changed. Figure 2.23 shows the effects of changing the *ta1* value to 0.25 and comparing it with the default value (0.5), and also shows the same comparison when the *ta1* value is changed to 4. Figure 2.24 shows the same comparison except that the variable is *ta2*. Figures 2.25 and 2.26 make the same comparison for the vertical channel, but the variables being analyzed are *tb1* and *tb2*. In the vertical channel the results are more pronounced.



SUMMARY:

BLOCK: AutoPilot

SUB-BLOCK(s): None.

INPUT(s): Horizontal and Vertical L.O.S. from Seeker Dynamics block,
 Missile/Captive Carry Velocity Magnitude from block Missile /Captive-Carry
 Dynamics.

OUTPUT(s): Commanded Horizontal and Vertical Acceleration to the Missile/Captive-Carry
 Dynamics block

AVAILABLE FUNCTION(s): None.

Figure 2.19. AutoPilot Model Block

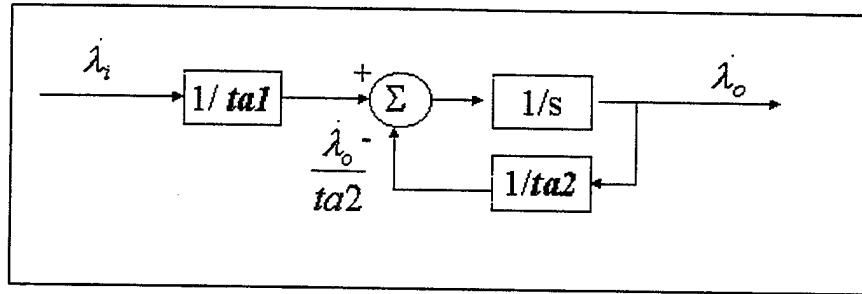


Figure 2.20. Block Diagram of the Commanded Horizontal Acceleration

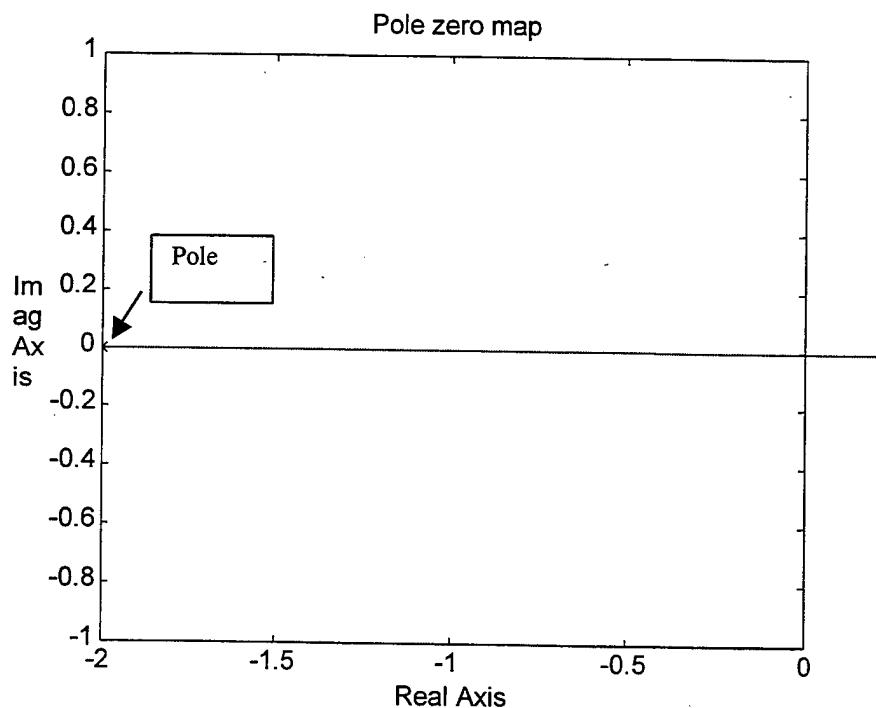


Figure 2.21. Pole-zero Diagram of the Commanded Horizontal Acceleration

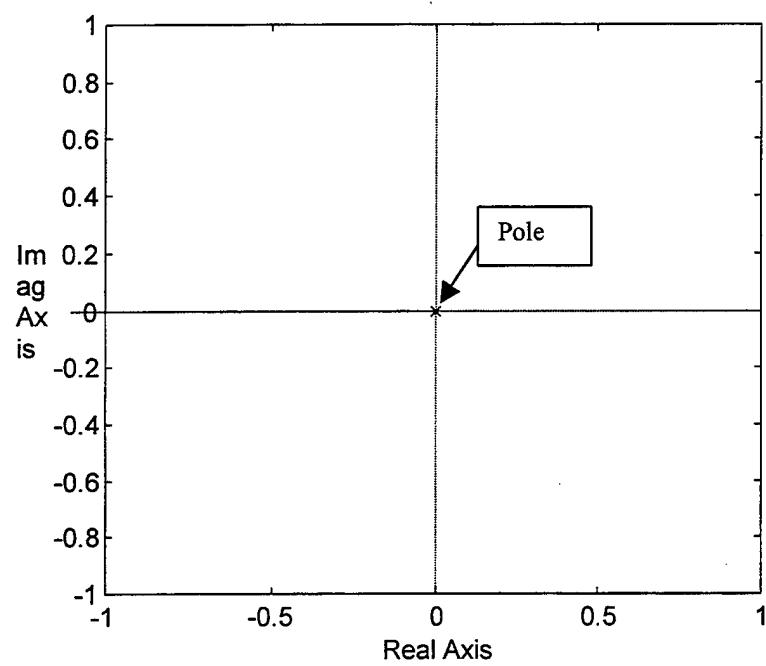


Figure 2.22. Horizontal Commanded Acceleration Root Locus

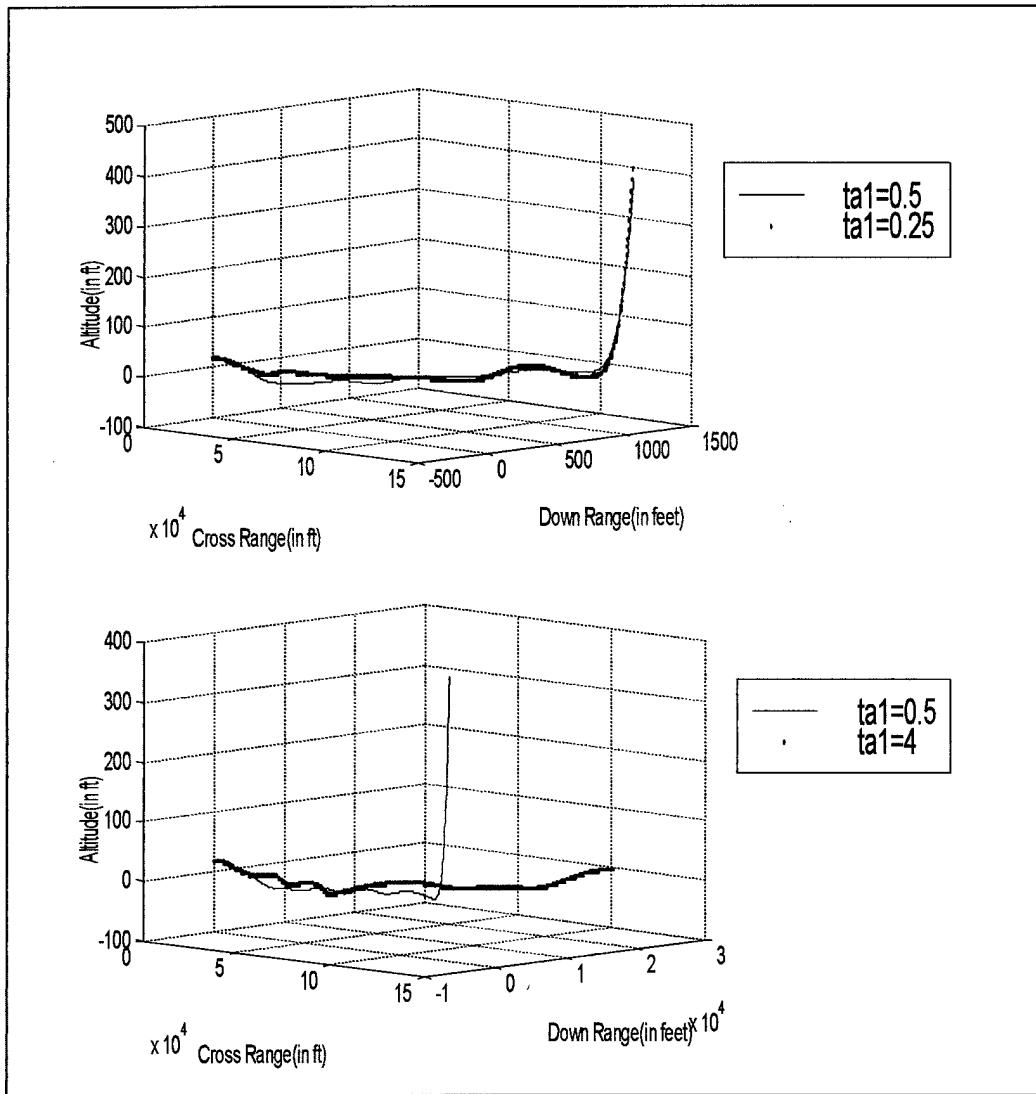


Figure 2.23. Missile Trajectory Effects Due to $ta1$ Variation

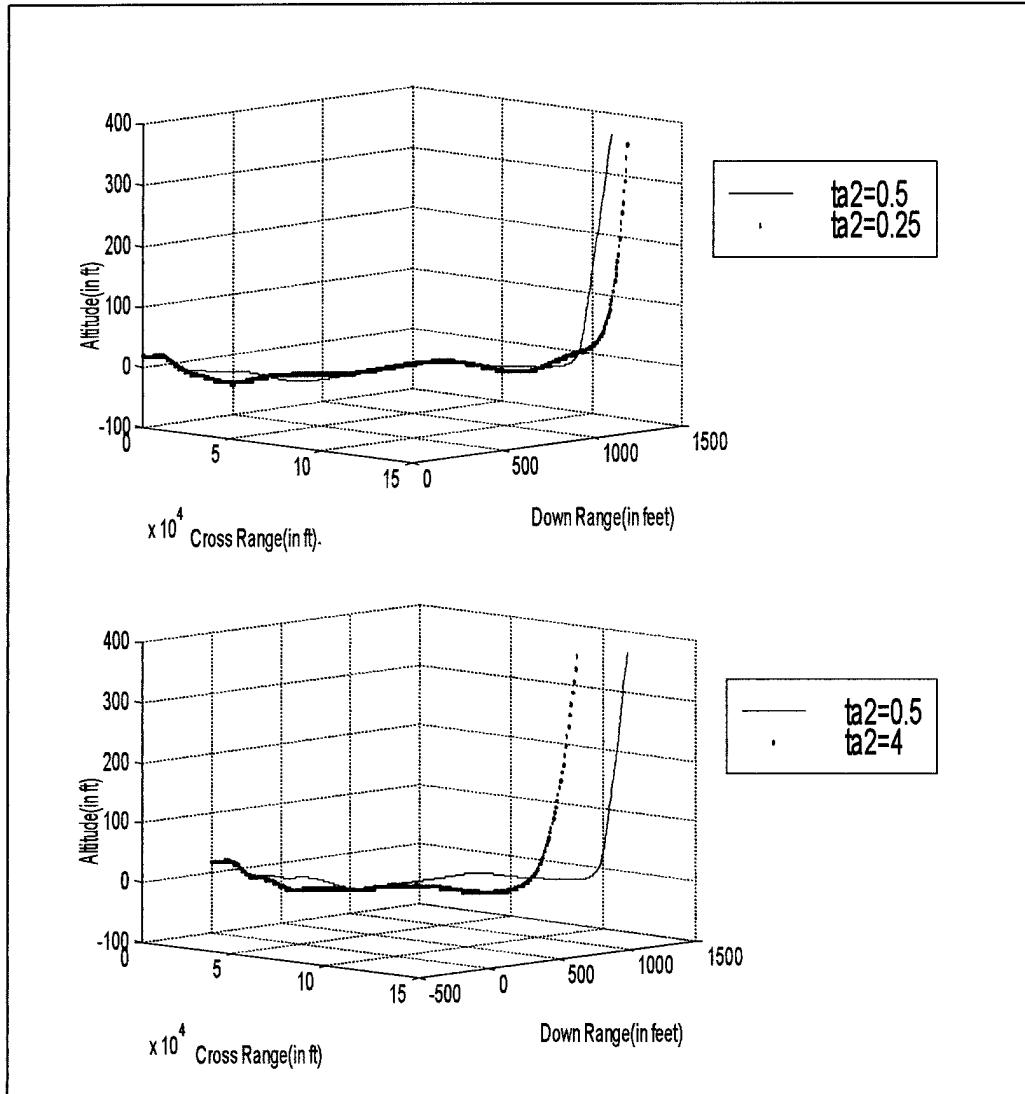


Figure 2.24. Missile Trajectory Effects Due to $ta2$ Variation

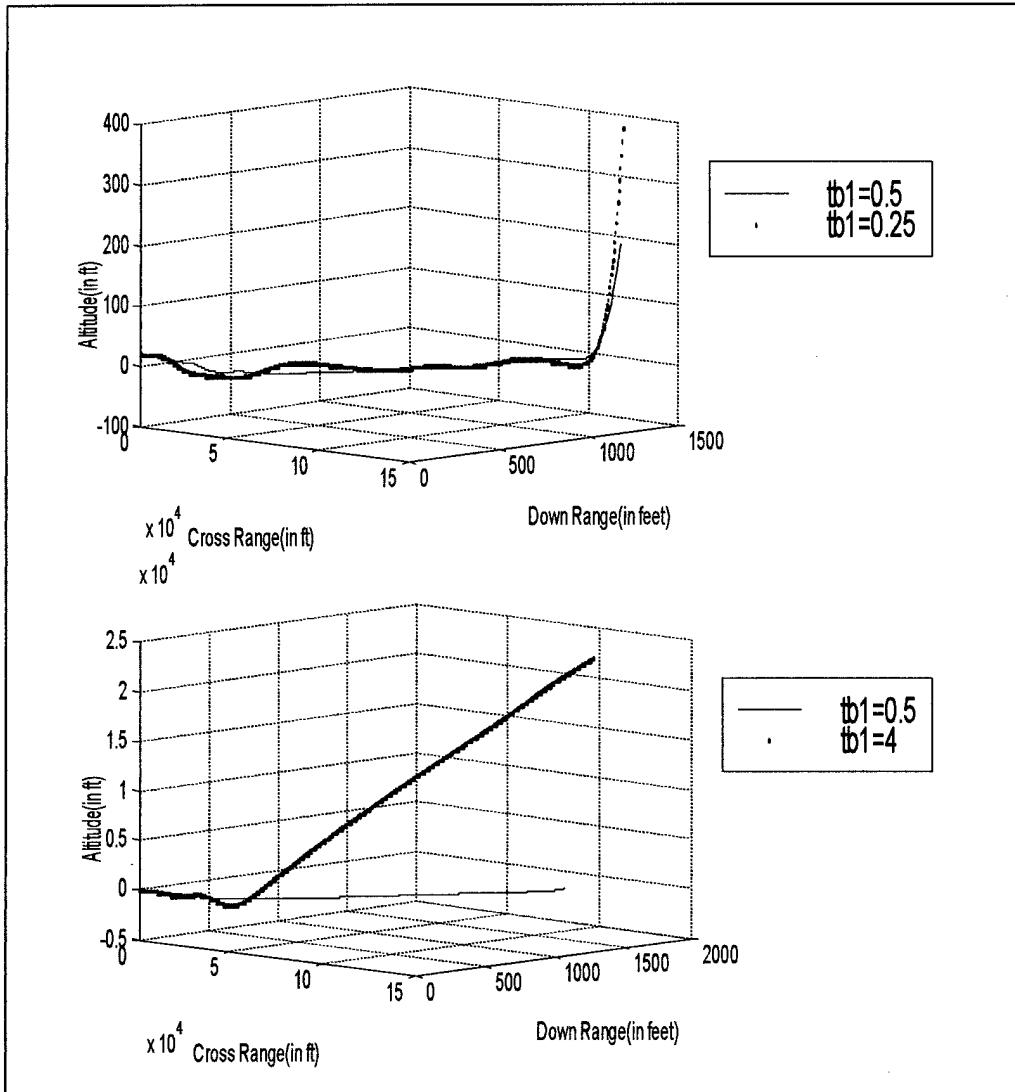


Figure 2.25. Missile Trajectory Effects Due to $tb1$ Variation

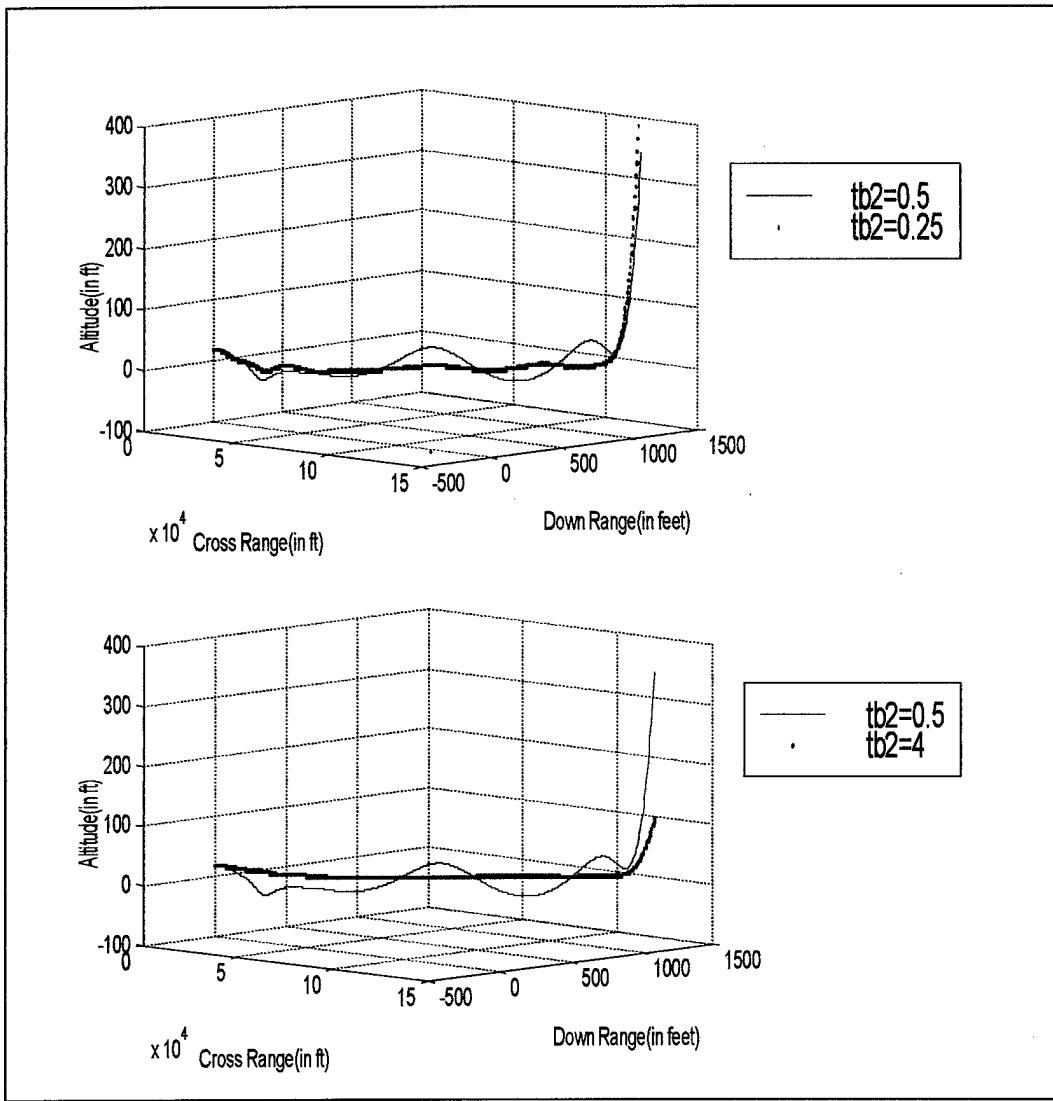


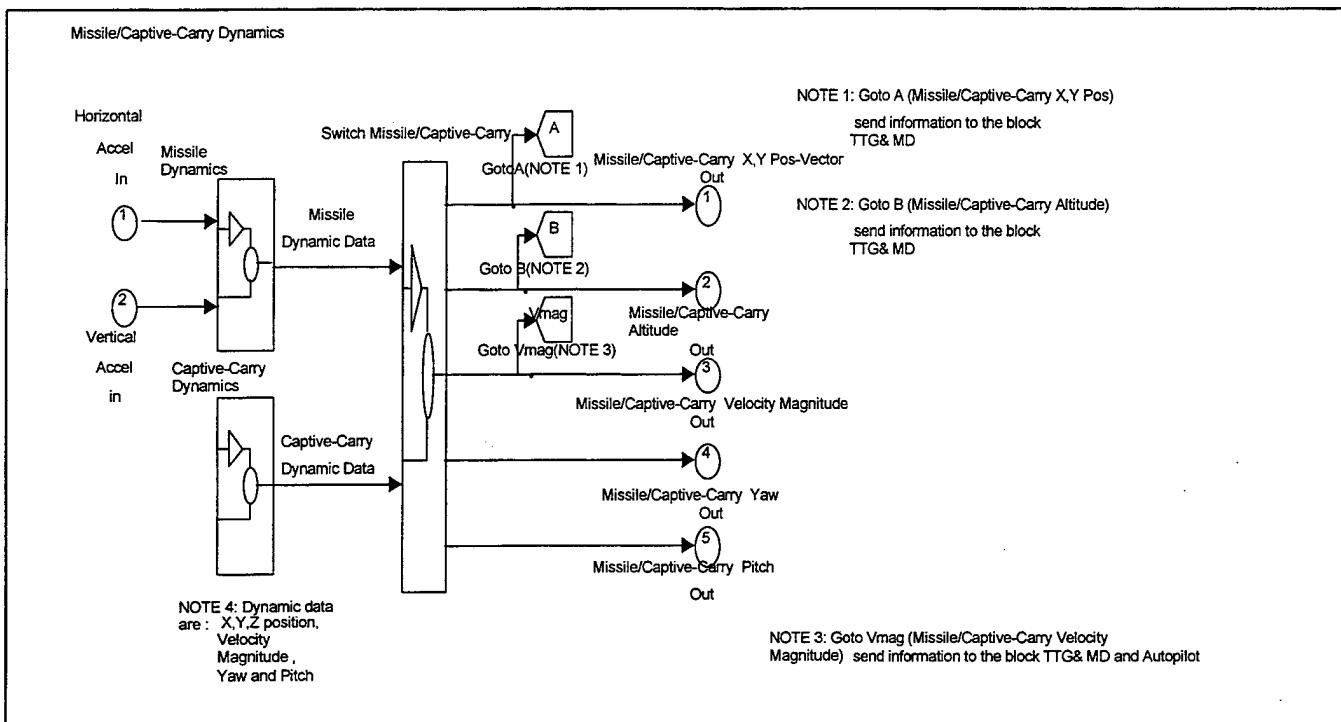
Figure 2.26. Missile Trajectory Effects Due to $tb2$ Variation

b. AutoPilot Description

The AutoPilot block is a 3 input block (Figure 2.19). It receives the horizontal and vertical rates from the Missile/Captive-Carry model and transforms them into accelerations by multiplying the angular rates (rad/sec) with the missile/captive-carry magnitude velocity (ft/sec). The results are accelerations in the horizontal and vertical channels that are sent to the Missile Dynamics. As a last comment about this block, it can be noted that, for both channels, there are delays between the acceleration rate generated by the missile seeker and the missile dynamic response.

3. Missile/Captive-Carry Dynamics

The Missile/Captive-Carry Dynamics block shown in Figure 2.27 is the system responsible for generating the threat dynamics (position, velocity magnitude, yaw and pitch) to the ASCM Digital model. The system is composed of three blocks: the Missile Dynamics, the Captive-Carry Dynamics and the Switch Missile/Captive-Carry. The system input is the commanded horizontal and vertical accelerations, but these are only used by the Missile Dynamics sub-block. There are three Simulink[©] GOTOs that send the selected threat position and magnitude velocity information to the TTG and Miss-Distance block.



SUMMARY:

BLOCK: Missile/Captive-Carry Dynamics

SUB_BLOCK: None.

INPUT(s): Horizontal and Vertical Accelerations from the AutoPilot Block.

OUTPUT(s): X, Y, Z threat (missile or captive-carry) position, threat velocity magnitude, Threat Yaw and Pitch.

AVAILABLE FUNCTION(s): None.

Figure 2.27. Missile/Captive-Carry Dynamics Block

The magnitude velocity is also sent to the AutoPilot block.

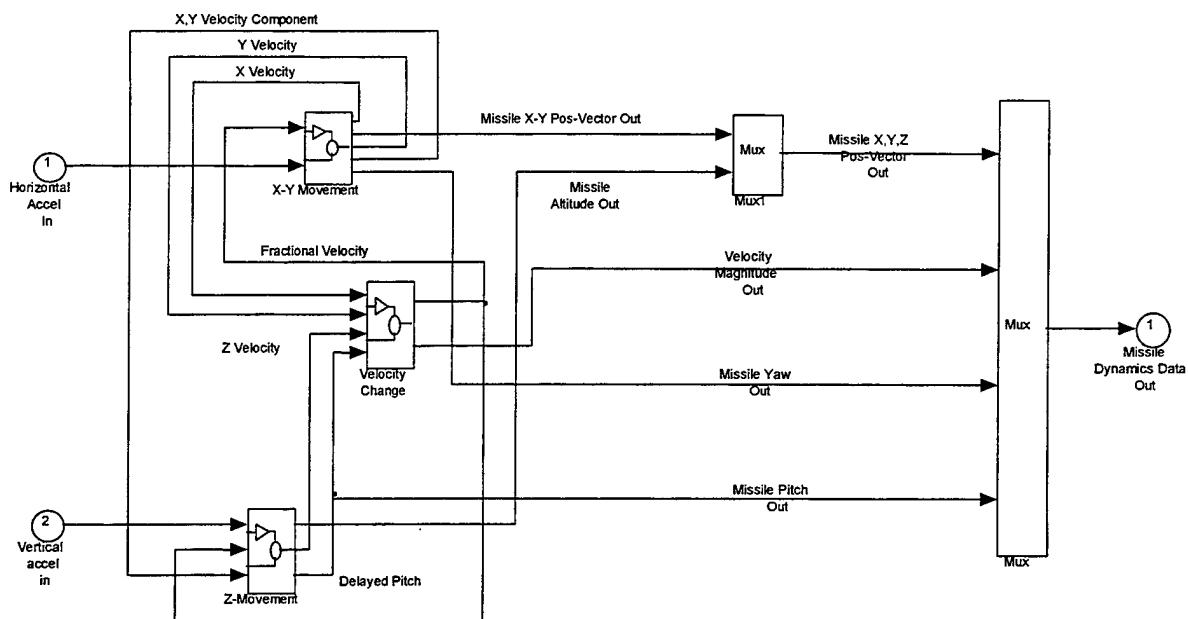
a. Missile Dynamics Analysis and Description

The Missile Dynamics model in Figure 2.28 takes the accelerations from the AutoPilot and calculates the next position of the missile. The conversion is accomplished using three sub-blocks: X-Y Movement, Velocity Change and Z-Movement. The three sub-blocks also calculate the pitch and yaw angles and the new missile velocity magnitude. Due to the inter-relationship between the three sub-blocks, the study of this block will be a little different compared with the previous ones. Here the model is described and an analysis of the effects of the missile dynamics variables tm (acceleration delay for X-Y movement), tmv (acceleration delay for Z movement) and tv (velocity change delay) in the missile trajectory are carried out.

The Missile Dynamics-Velocity Change sub-block is shown in Figure 2.29. This sub-block receives four inputs: the X, Y and Z velocity components from the X-Y and Z Movement sub-blocks, respectively, and the pitch information also from the Z Movement sub-block. The three velocity components are combined in a vector by a multiplexer and from there the velocity magnitude is obtained. The velocity magnitude is further adjusted by Changed Velocity Magnitude sub-block taking into account the missile pitch information as

$$vel = 1322 - ((pitch - 0.01) \times 575) \quad (21)$$

Missile Dynamics



SUMMARY:

BLOCK: Missile Dynamics

SUB-BLOCKS: X-Y Movement
Velocity Change
Z Movement

INPUT(s): Horizontal and Vertical Acceleration from the AutoPilot block.

OUTPUT(s): Missile dynamic data (position, velocity magnitude, yaw, and pitch).

AVAILABLE FUNCTION(s): None.

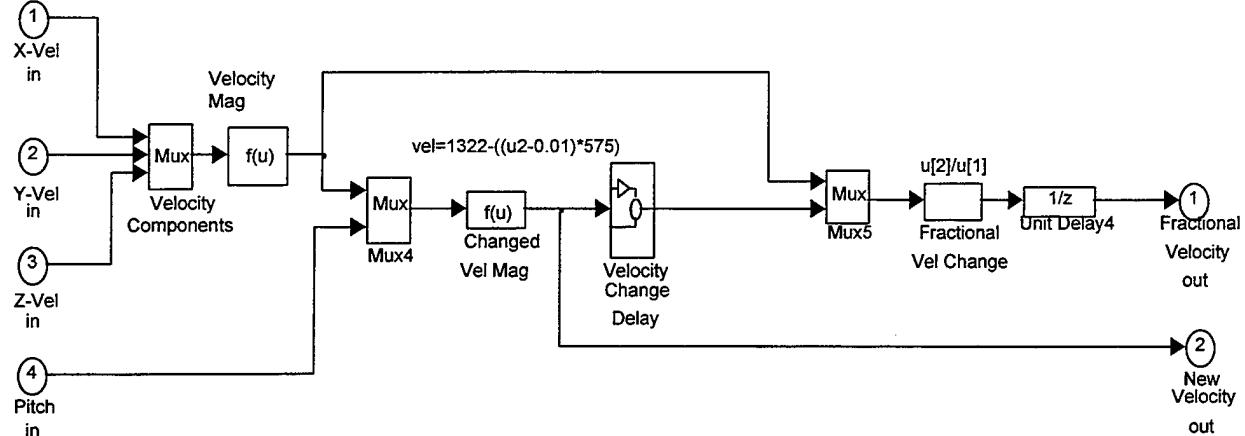
Figure 2.28. Missile Dynamics Block

where the value 1322 is the missile speed in ft/sec. The negative component of Eq. 21 is a correction factor due to pitch induced drag.

The updated velocity magnitude is sent to the Missile Dynamics block and to the Velocity Change Delay sub-block (missile inertia). After the delay, the fractional velocity is calculated by dividing the updated velocity due to the pitch. The fractional velocity change is sampled and delayed and represents the delay between the velocity calculation for each coordinate and its effective use by the missile components responsible for its repositioning. The Velocity Change Delay block diagram is shown in Figure 2.30 and has a forward and feedback gain. The closed-loop transfer function is a first order system with one pole at $-tv$, and using the default value of $tv=0.5$, the pole is at -0.5 . The larger is the value of tv , the larger the missile inertia becomes. Figure 2.31 shows the missile trajectory for $tv=0.25, 4.0$ and 0.5 (default).

The second sub-block, the X-Y Movement sub-block, has two inputs and is shown in Figure 2.32. One is the fraction velocity generated in the block Velocity Change and the other is the horizontal acceleration from the AutoPilot block. These two inputs generate the missile X and Y velocity components, the missile X-Y position and the missile yaw.

Missile Dynamics
Velocity
Change



SUMMARY :

BLOCK : Missile Dynamics – Velocity Change

SUB-BLOCK(s): Velocity Change Delay

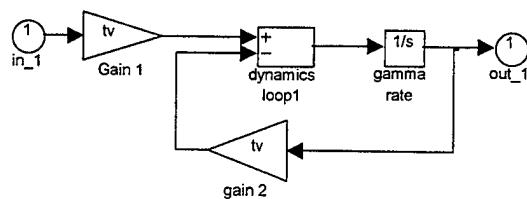
INPUT(s): X-Velocity, Y-Velocity from block X-Y Movement,
 Z- Velocity ,and
 Delayed pitch from block Z – Movement.

OUTPUT(s): Fractional Velocity to the blocks X-Y Movement and Z- Movement, and
 Velocity Magnitude to the block Missile Dynamics.

AVAILABLE FUNCTION(s): None.

Figure 2.29. Velocity Change Sub-Block

Velocity Change Delay



SUMMARY:

BLOCK: Velocity Change Delay

SUB-BLOCK(s):None

INPUT(s): Changed magnitude velocity from block
Velocity Change.

OUTPUT(s): Velocity delayed

AVAILABLE FUNCTION(s): None.

Figure 2.30. Velocity Change Delay Sub-Block

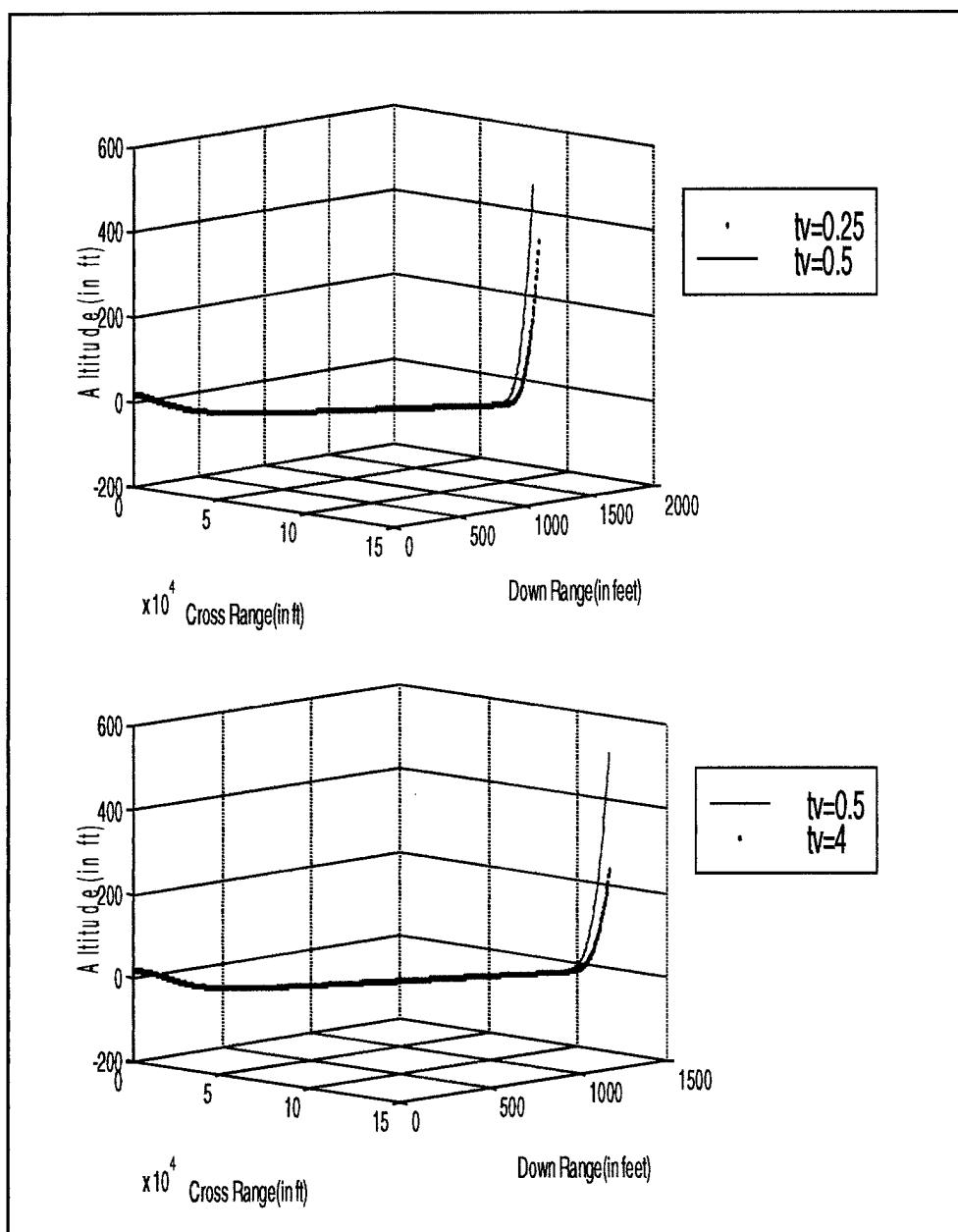
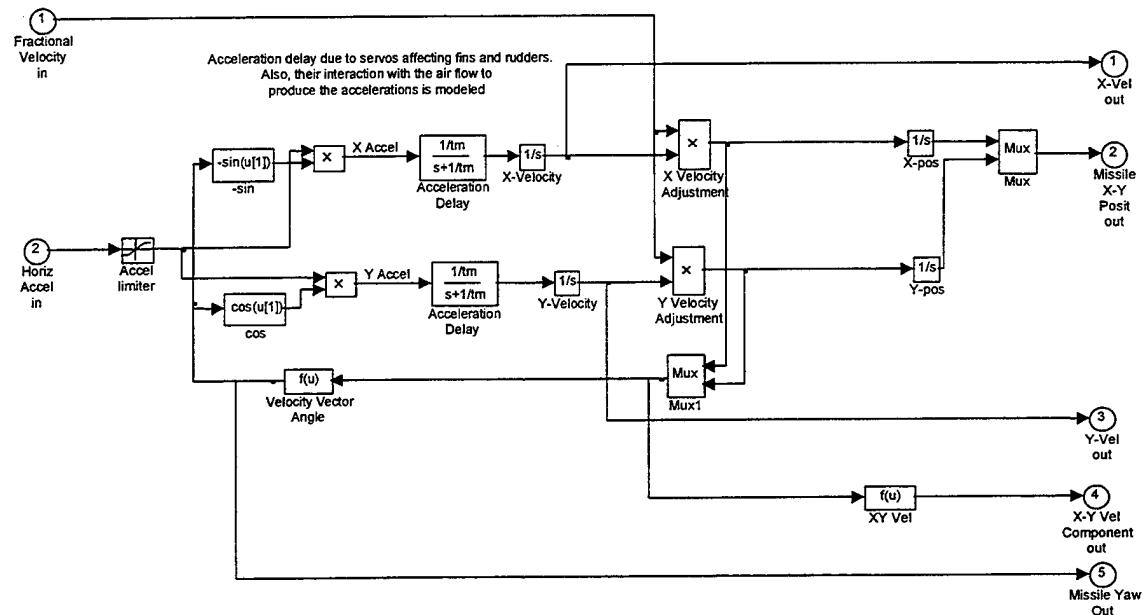


Figure 2.31. Missile Trajectory Effects Due to tv Variation

Missile Dynamics
X-Y Movement



SUMMARY :

BLOCK: Missile Dynamics - X-Y Movement

INPUT(s): Fractional Velocity from block Velocity Change,
Horizontal Acceleration from Missile Dynamics block.

OUTPUT(s): X- Velocity, Y- Velocity to the block Velocity Change,
X-Y Velocity component to the block Z-Movement,
Missile X-Y position and missile yaw to the block Missile Dynamics.

AVAILABLE FUNCTION(s): None.

Figure 2.32. Missile Dynamics X-Y Movement

The horizontal acceleration is introduced in the block and passed by the Acceleration limiter block that limits the received horizontal acceleration between ± 130 ft/sec² and prevents the model from receiving accelerations not possible in the real world. The horizontal acceleration is then decomposed into its X and Y components by multiplying it with the sine and cosine of the current yaw. Both acceleration components are delayed by a first order system whose gain is $1/tm$ and its pole is at $-1/tm$. These delays are due to the servos affecting the fins and rudders and also the interaction between fins and rudders with the airflow.

After the acceleration components being delayed, they are integrated to generate the missile X and Y velocity components used in the Velocity Change block. They are then multiplied by the fractional velocity and thus generate the fractional velocity in those directions and give the velocity increments required to reposition the missile towards the target. These increments are used in two ways. The first is to integrate them into a vector to get the new missile X-Y position that is sent to the Missile Dynamics block. The second is to combine them in a 2x1 vector. From that vector, the angle between them is obtained, which is the missile yaw, and the vector magnitude is the missile X-Y velocity magnitude component used in the Z-movement block. The calculated yaw is used as the output to the Missile Dynamics block and to get the acceleration components in the X-Y direction. The effects of tm in the missile trajectory can be understood without its root locus due to the similarities between tm and the AutoPilot delays. The larger the tm value the less the servo delays. The smaller the tm value the larger the delay. The same conclusion can be extended to the delay tmv in the Z

Movement sub-block. Figure 2.33 shows the effects in the missile trajectory using $tm=1$, 8 and 2(default).

The Z Movement or Altitude of the Missile Flight Simulation sub-block receives three inputs: the vertical acceleration from the AutoPilot block, the fractional velocity from the Velocity Change sub-block and the X-Y velocity component from the X-Y Movement block. The vertical acceleration passes through a acceleration limiter as described before and, after that, it has its value multiplied by the cosine of the current pitch, which gives the acceleration value in the Z direction. The acceleration in the Z direction is delayed due to servos by a value $1/tmv$. The delayed acceleration is then integrated and generates the velocity Z component. The velocity Z component is multiplied by the fractional velocity calculated in the Velocity Change sub-block giving the adjusted velocity in the Z direction.

The adjusted velocity is integrated again and the missile altitude is obtained. The adjusted velocity gives the Z velocity component which, combined with the X-Y velocity component in a vector by the Simulink[©] multiplexer, also gives the missile current pitch calculated from the angle between them. The pitch angle is used to get the vertical acceleration component in the Z direction and, after being delayed, it is sent to the Missile Dynamics block and to the Velocity Change sub-block. The effects of tmv in the missile trajectory shown in Figure 2.34 are the same effects of tm for the X-Y direction. The larger the tmv value the less the delay, the smaller the tmv value the larger the missile delay in the Z direction.

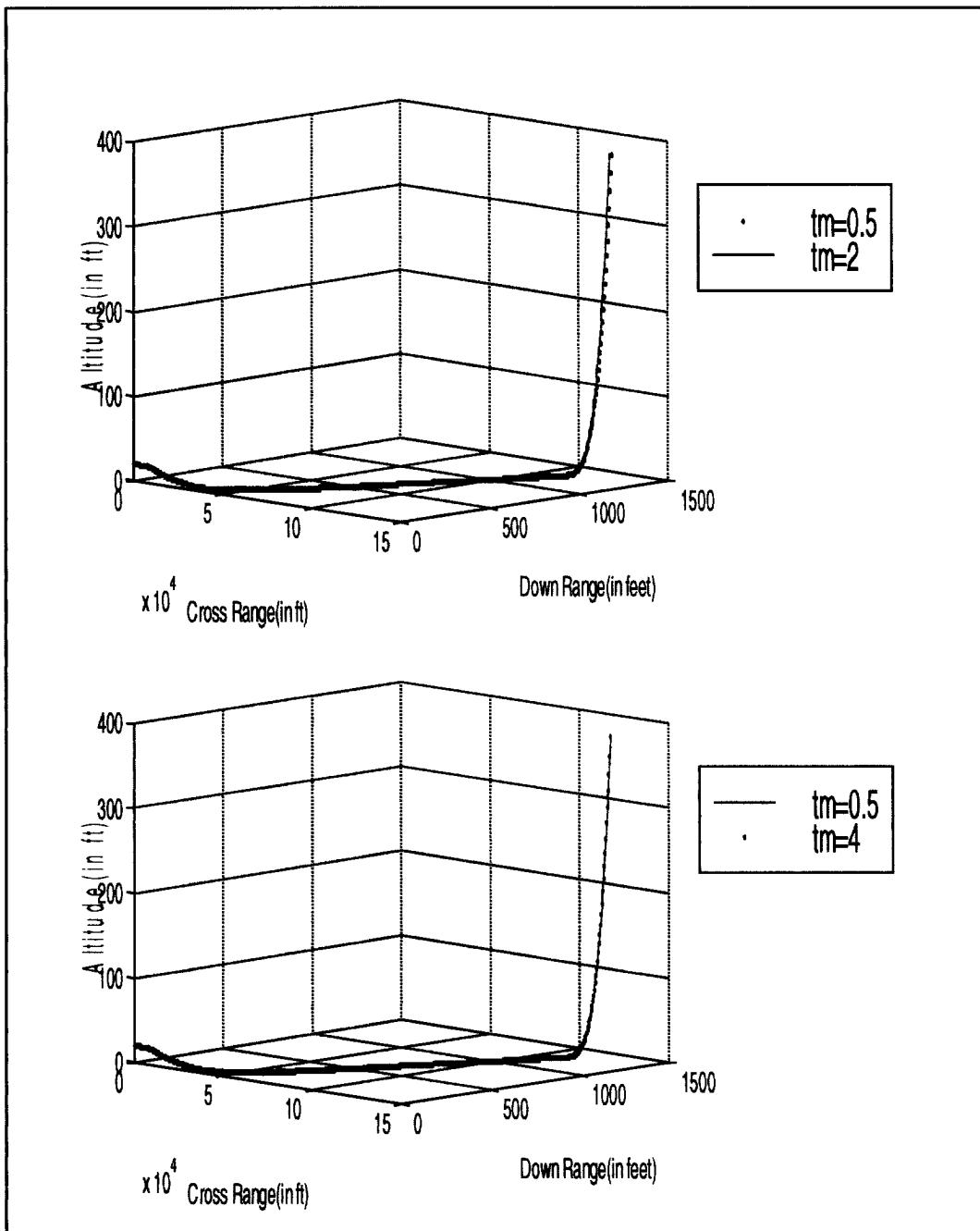


Figure 2.33. Missile Trajectory Effects due to tm Variation

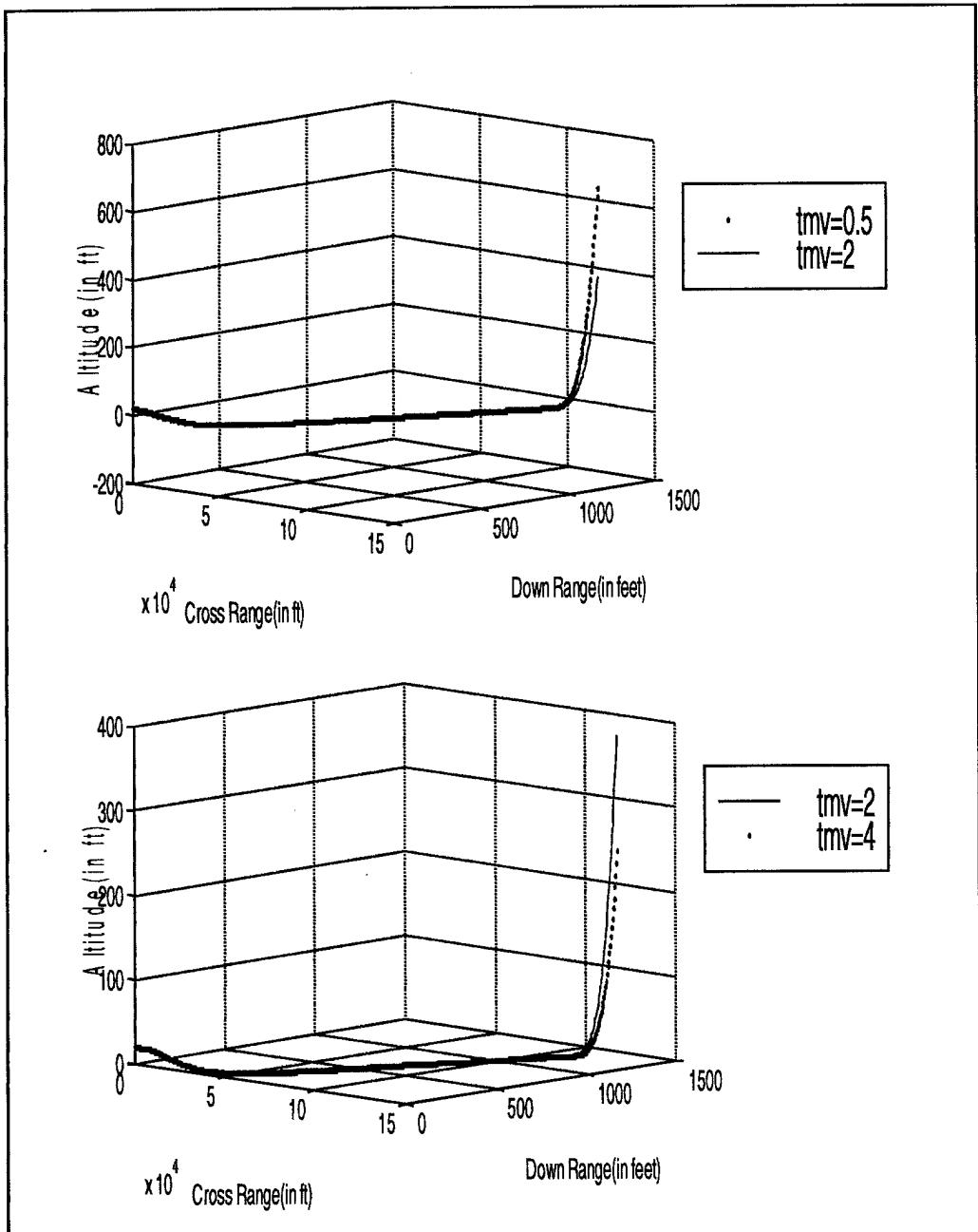


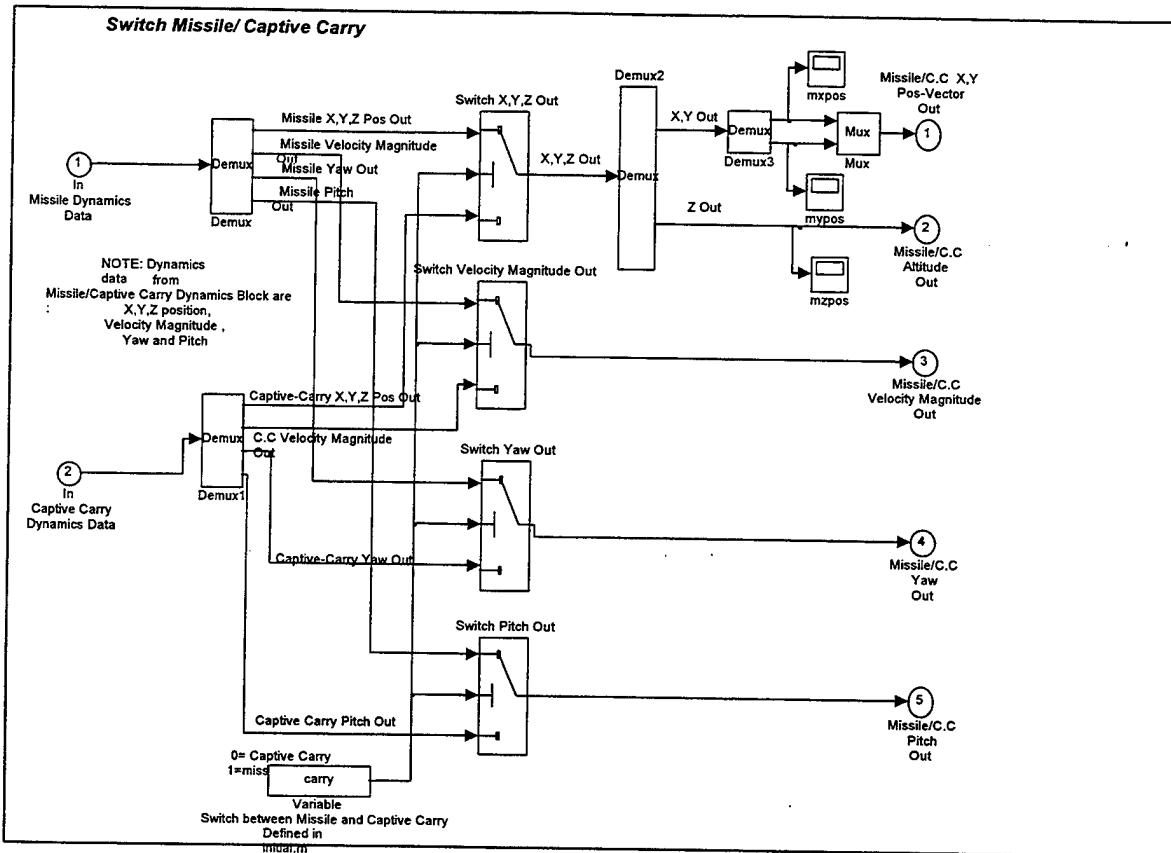
Figure 2.34. Missile Trajectory Effects due to tmv Variation

b. ***Switch Missile/Captive-Carry***

The Switch Missile/Captive-Carry block is shown in Figure 2.35. This block selects what threat configuration (missile or captive-carry) is available to the ASCM Digital model block. The dynamics data input are the threat position, the velocity magnitude of the platform and the yaw and pitch angles assumed by the platform during its trajectory toward the target.

The input missile or captive-carry dynamics is switched by a threshold (*carry*) that is defined in the initialization code *intial.m*. If the value of *carry* is one, the selected threat is the missile. If the *carry* value is zero, the simulation is run with the captive-carry configuration.

The transmission of dynamics data to the Missile/Captive-Carry Dynamics block is straightforward for the threat velocity magnitude, yaw and pitch information. The threat position information is decomposed in the X-Y and Z position components to fit the format required by the Missile/Captive-Carry Dynamics and ASCM Digital Model blocks. Before sending the X-Y threat position, it is decomposed again in order to allow this block to send the selected threat position components to the MATLAB workspace using the Simulink[©] scopes *mxpos*, *mypos* and *mzpos* that generate variables with the same name in the workspace.



SUMMARY:

BLOCK: Switch Missile/Captive-Carry

SUB-BLOCK(s): None.

INPUT(s): Missile dynamic data (position, velocity magnitude, yaw, and pitch) and Captive-Carry dynamic data (position, velocity magnitude, yaw, and pitch) from the block Missile / Captive-Carry Dynamics.
 Switch **carry** allows selection between the dynamics outputs of the missile or the captive- carry.

OUTPUT(s): Selected threat (missile or captive-carry dynamic data -position, velocity magnitude, yaw, and pitch).

AVAILABLE FUNCTION(s): None.

Figure 2.35. Switch Missile/Captive-Carry Dynamics Block

4. Time_to_Go (TTG) and Miss_Distance

The Time_to_Go (TTG) and Miss_Distance block is shown in Figure 2.36 and performs several important tasks during a simulation. The most important is to stop the simulation when the missile hits the target or the captive-carry aircraft crosses over the target. The other tasks are to calculate the distance between the threat and target, and to send this information to the MATLAB[©] workspace for the miss distance calculation, and to also generate an internal parameter that warns when the time-to-go is equal to or less than the pre-set value defined in the *initial.m* code through the variable *ttg*.

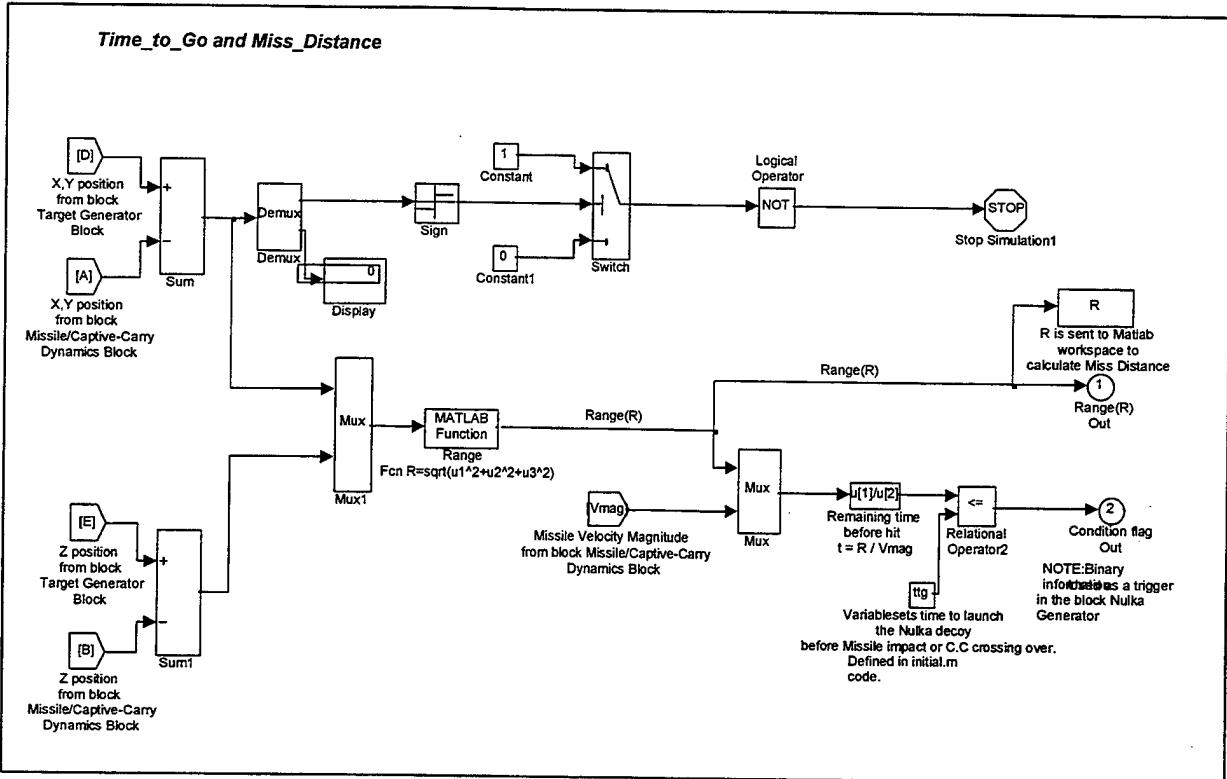
The TTG and Miss_Distance block receives as inputs the X-Y and Z coordinates of the threat and the threat velocity magnitude from the Missile/Captive-Carry Dynamics block, and the selected target from the Target Generator block. The threat and target positions are received in the X-Y and Z position channels (horizontal and vertical channels, respectively). In both channels, the missile position is always subtracted from the target position because the former is smaller than the later in the horizontal plane and thus generates initially a positive position difference in the horizontal channel and a negative position difference in the vertical channel.

The horizontal position difference takes two different paths. In the first path, the horizontal position difference is separated in its X and Y components. The X position difference is introduced to the Simulink[©] block *sign* to detect when the X position difference becomes negative thus signifying that the target was already hit or the captive-carry aircraft is crossing over. This signal variation of the X position difference is compared with a threshold, equal to zero (logical switch). Before the missile reaches the

target, the X position difference is always positive which makes the *sign* block send a value of one to the logical switch. The logical switch compares that value with the threshold and allows the constant value of one in port one to be released to the logical port “NOT”. The logical port “NOT” transforms the unity value in zero, which is sent to the Simulink *stop* block that permits the simulation to continue. When the missile hits or the captive-carry aircraft crosses over the target, the X position the difference has a negative value and the *sign* block sends a negative unity value to the logical switch that compares it with the threshold value and releases the zero value to port 3. The “NOT” function transforms the value from one to zero to stop the simulation. In the second path, the X-Y position differences are combined in a 2x1 vector with the Z position difference. The difference vector is introduced to a function that calculates the magnitude of the vector (range).

The range value also takes two paths. One goes to the section that sends its values to the MATLAB workspace through the variable *R* for further and precise miss distance calculation and also to the block ASCM/Captive-Carry model in order to display the approximate miss distance at the end of the simulation. The second path divides each *R* value by the threat velocity magnitude from the Missile/Captive-Carry Dynamics block and, to give the remaining time before the missile hits the target or the captive-carry aircraft crosses over the target. This value is compared with *ttg* by the relation operator and when its value is less than the time-to-go selected for the simulation, the relation operator sends a positive unity value to the ASCM/Captive-Carry model, which is called the internal parameter *flag*.

The internal parameter *flag* in the ASCM/Captive-Carry model is used to mark the exact moment the target to launch the Nulka decoy. The Nulka Generator and the Switch target/Nulka blocks use the parameter *flag*.



SUMMARY:

BLOCK: Time_to_Go and Miss_Distance

SUB-BLOCK(s): None.

INPUT(s): Selected target position
Selected threat position and
Threat velocity magnitude

OUTPUT(s): Signal stop that stops the simulation,
Variable **R** (range) sends information to the Matlab
Internal parameter **flag**

AVAILABLE FUNCTION(s): None.

Figure 2.36. Time_to_Go (TTG) and Miss_Distance Block

III. GENERAL TARGET MODEL

The General Target Model in Figure 3.1 is the block responsible for generating the target that is projected to the threat seeker. In the ASCM Digital Model, there are three types of targets. The first two targets represent a surface combatant. They are called Original Target and Ship. The last one is a Nulka decoy launched from one of the two surface combatants. The two surface combatants are generated in the Target Generator sub-block shown in Figure 3.2. The Nulka decoy is generated in the Nulka Generator block shown in Figure 3.3.

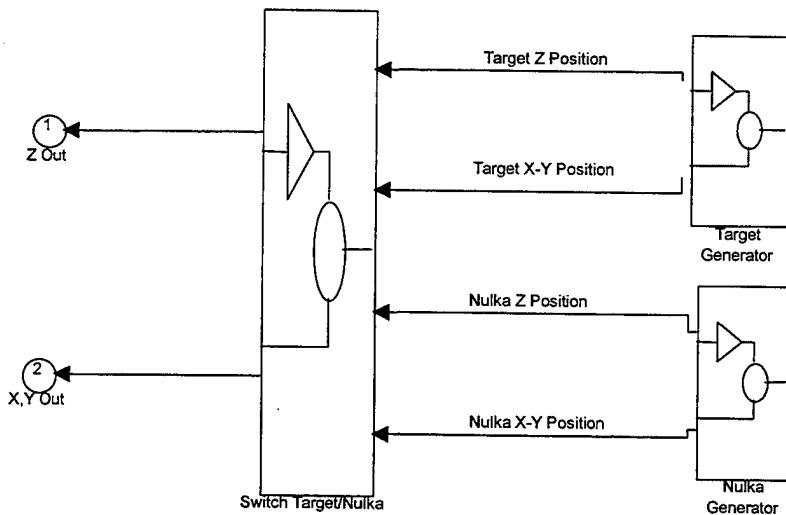
The General Target model block also has the sub-block Switch Target/Nulka shown in Figure 3.4. It has the ability to receive the selected target (Original Target or Ship) and the Nulka positions and to select the one to be projected to the threat seeker in the ASCM Digital Model. The parameter used for this decision is a function of the internal parameter *flag* generated in the TTG and Miss_Distance block.

A. TARGET GENERATOR

The Original Target and Ship sub-sub-blocks within the Target Generator sub-block are used to generate the type of target desired for the simulation. This decision is a function of the variable *swtsh*, defined in the initialization code before running a simulation. In order to avoid confusion, the Original Target will be called the first configuration and the Ship Target will be called the second configuration.

The first configuration is a ship in course 000° (assuming the Y axis as the true North) at a speed of 12 knots at a distance 120,000 feet from the inertial origin. This first configuration is called Original Target because it was the first target developed for the

General Target Model



SUMMARY:

BLOCK: General Target Model

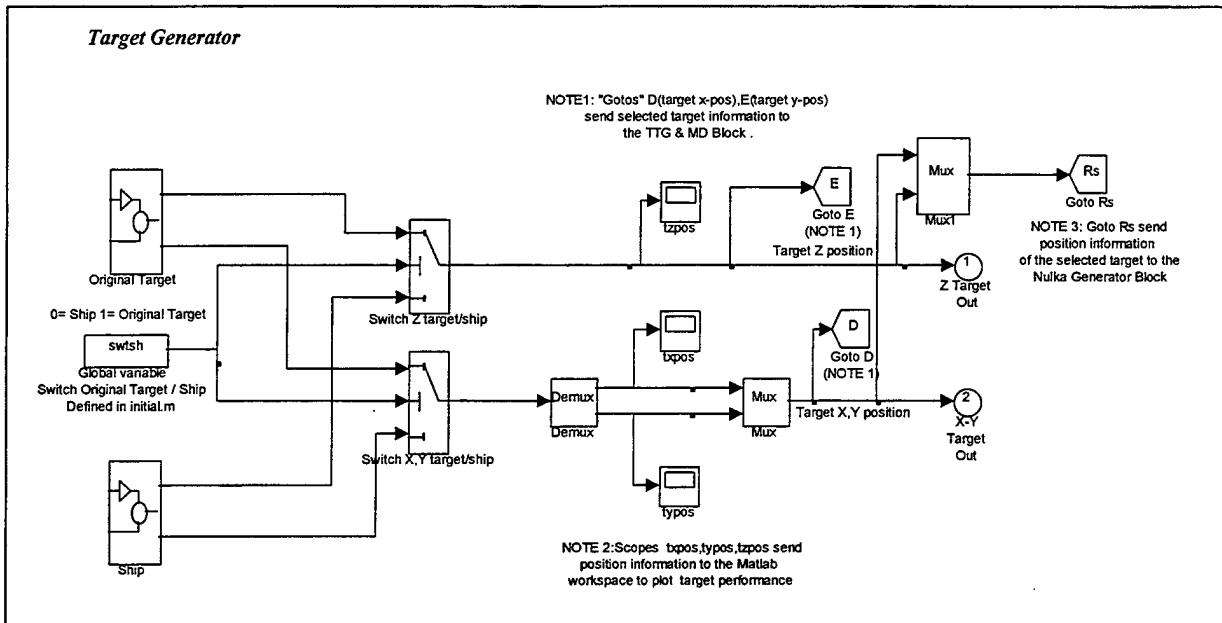
SUB-BLOCK(s): Target Generator
Nulka Generator
Switch Target/Nulka

INPUT(s): None.

OUTPUT(s): Selected threat position to the ASCM Digital Model.

AVAILABLE FUNCTION(s): None.

Figure 3.1. General Target Model



SUMMARY :

BLOCK: Target Generator

SUB-BLOCK(s): Original Target
Ship

INPUT(s): **swtsh** (switch-to-ship) –general switch that allows selection between the the Original Target (heading 000° at 12 knots) or Ship (heading 000° at 10 knots).

OUTPUT(s): Three Simulink© GOTOs are used to send information to other blocks:

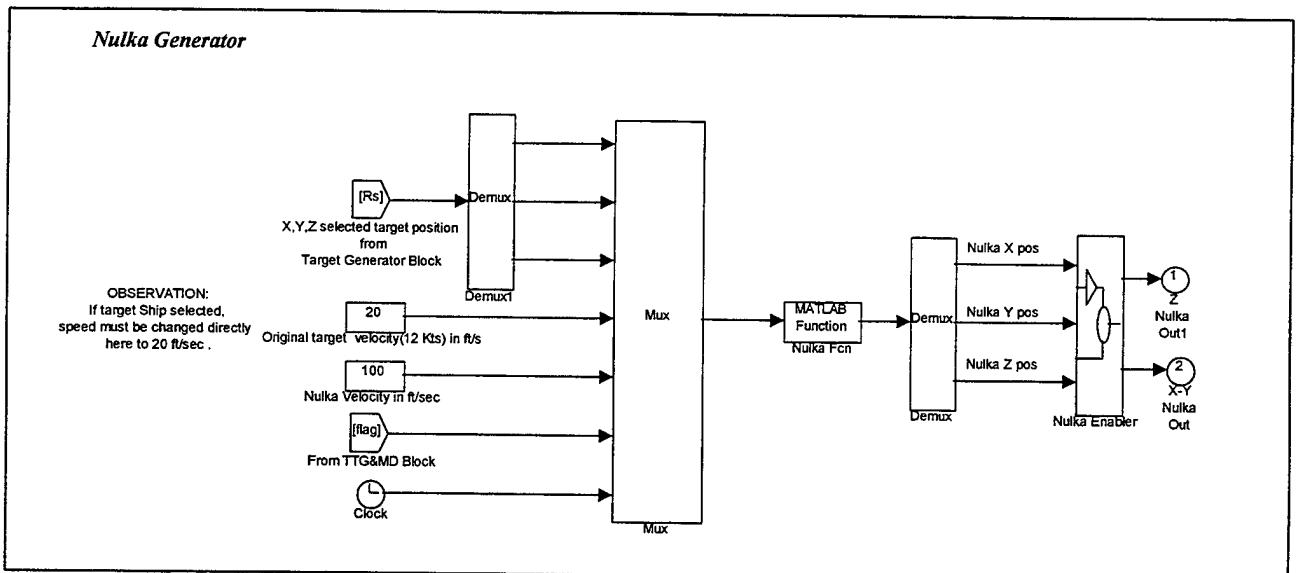
- GOTOs **D** and **E** send the selected target position to the “Time_to_Go and Miss_Distance” block.

- GOTO **Rs** send position information to the Nulka Generator block.

- Also, the selected target information is sent to the General Target Model.

AVAILABLE FUNCTION(s): None.

Figure 3.2. Target Generator Sub-Block



SUMMARY:

BLOCK: Nulka Generator

SUB-BLOCK(s): Nulka Enabler

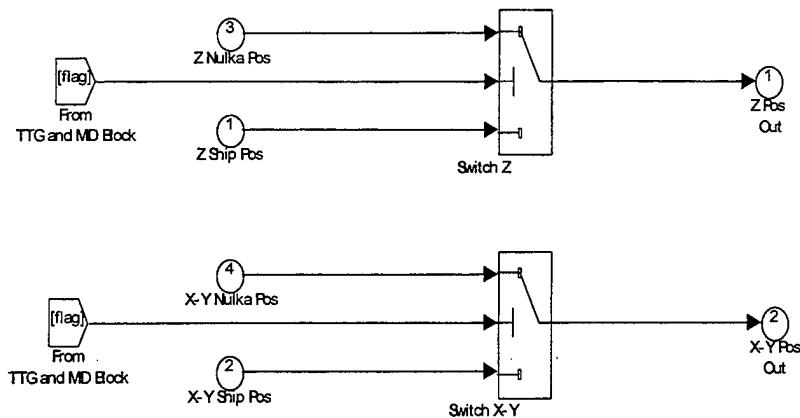
INPUT(s): FROM *Rs* (selected target position from Target Generator block),
Ship velocity 20 ft/sec or (50/3) ft/sec,
Nulka velocity,
FROM *flag* internal parameter, received from the block Time_to_Go and
Miss_Distance,
Current simulation time

OUTPUT(s): Nulka position to the General Target Generator block.

AVAILABLE FUNCTION(s): *nulka.m* (See Appendix D).

Figure 3.3. Nulka Generator Sub-Block

Switch Target/Nulka



SUMMARY:

BLOCK: Switch Target/Nulka

SUB-BLOCK(s): None.

INPUT(s): Selected target position
Nulka position
Internal parameter *flag* from the TTG and Miss_Distance block

OUTPUT(s): The general target position to the ASCM Digital Model.

AVAILABLE FUNCTION(s): None.

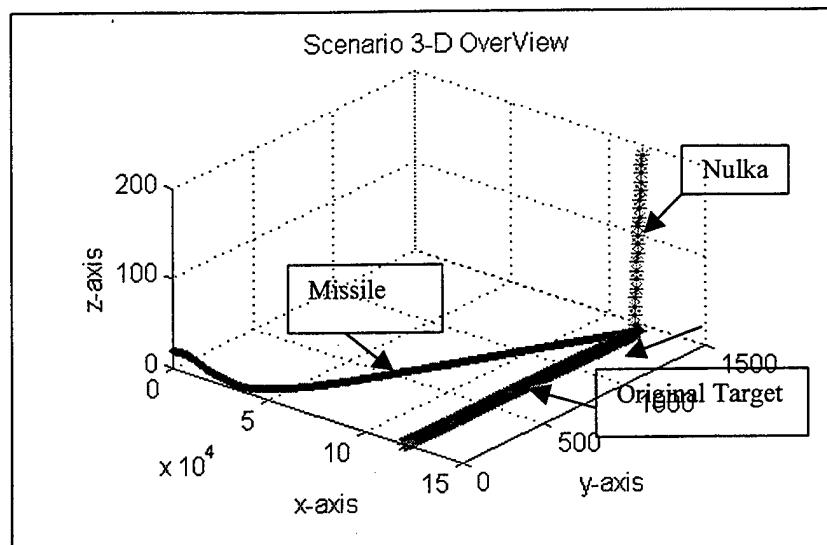
Figure 3.4. Switch Target/Nulka Sub-Block

ASCM Digital model before improvements were made. The simulations run for this thesis all were assumed to have the Original Target as the primary target due to its tactical position with respect to the incoming missile.

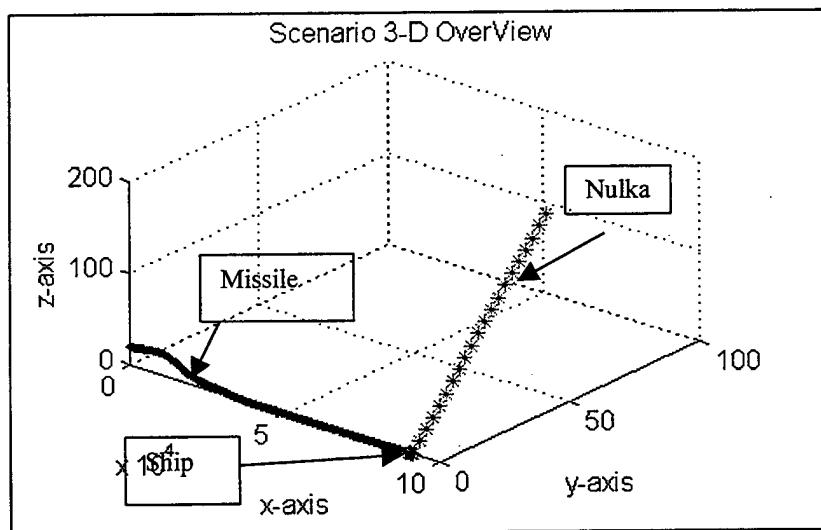
The second configuration is a target moving in the X direction on a course 090° at a speed of 10 knots, at a distance of 90,000 feet from the inertial origin. It was implemented in order to offer the user an option to observe what the threat seeker responses are for these two types of targets. Figure 3.5 presents the two target profiles available.

The Target Generator sub-block in Figure 3.2 generates the X, Y and Z coordinates of the two types of target configurations (Original Target and Ship), and introduces them to two switches, one for the Z coordinate and the other for the X-Y coordinates. These switches have the threshold values equal to one and, depending on the variable *swtsh* value, (*swtsh*=1 for the Original Target and *swtsh*=0 for Ship) they allow the Original Target or Ship coordinates to pass to the Switch Target/Nulka sub-block. After one target is selected, which in our case is always the Original Target, its position is sent by the scopes *txpos*, *typos* and *tzpos* to the MATLAB© workspace to be stored in the structure *store* and also to be used for simulation graphing as necessary.

There are two Simulink© GOTOs that send the selected target position to the TTG & MD block shown in Figure 3.2 in order to give that block the necessary target information to stop the simulation when the threat hits or crosses over the target on the sea surface. The selected target position is also important to the Nulka Generator block



(a)



(b)

Figure 3.5. Target Profiles Available in the ASCM Digital Model: (a) Original Target and (b) Ship

in order to give an initial position to the Nulka rocket before launching. The GOTO *Rs* sends that information to the Nulka generator block.

1. Original Target Generator

The model presented in Figure 3.6 shows three distinct sources that generate the Original Target X, Y and Z coordinates. The Simulink[©] block that generates the Z coordinate to the Target Generator block sends constant values of zero which means that the target has no altitude. The X coordinate is also generated for a block that always sends a constant value for the target X coordinate equal to 120,000 feet (or 36.6 km) from the inertial origin. Finally, the Y coordinate is generated by a Simulink[©] *ramp* function whose slope is given as a function of the target velocity. For a target at 12 knots, the slope corresponds to 20 ft/sec. The resultant course is 000°. The X and Y coordinates are combined together by a Simulink[©] multiplexer block to be sent in a 2x1 vector format plus the Z coordinate to the Target Generator block.

2. Ship Generator

The Ship Generator block in Figure 3.7 has the same configuration as the Original Target except that the ramp function generates the target position in the X direction with a slope of $(50/3)$ ft/sec which represents a target at a speed of 10 knots. The Y and Z directions are set to zero causing no displacement in those directions. The resultant course is 090°.

B. NULKA GENERATOR

The Nulka Generator block shown in Figure 3.3 and is the model of an active expendable decoy system developed for ships to face multiple threats in short reaction times encountered in several operational areas, but particularly in littoral waters.

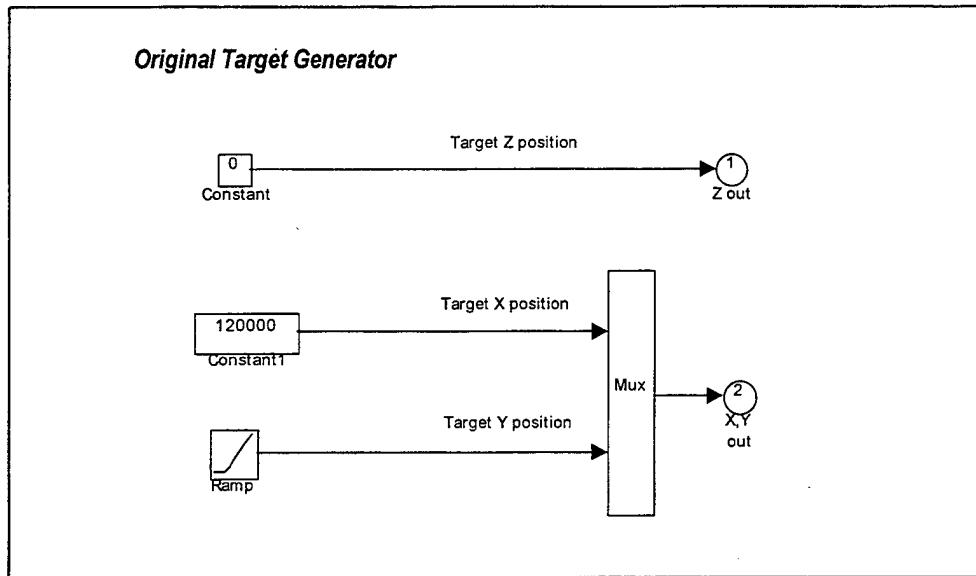


Figure 3.6. Original Target Generator Model

The system is composed basically for a fire control system, a launcher and the decoy itself as shown in Figure 3.8. The decoy consists of a maneuvering rocket and a microwave transponder payload used to seduce the threat in its final approach to the ship. The Nulka system allows for setting the decoy flight path and speed. The Nulka system accounts for the ship's motion and the wind conditions. The Nulka rocket positions the transponder payload in the direction of the threat in order to maximize the decoy position for the missile seduction. The microwave payload generates sufficient effective radiated power (ERP) to seduce the missile by offering to the seeker a larger radar cross section than the ship radar cross section. Nulka, compared with chaff, has the advantage of being unaffected by the weather conditions and does not require any ship maneuvers. Also Nulka can afford 360° of azimuth coverage.

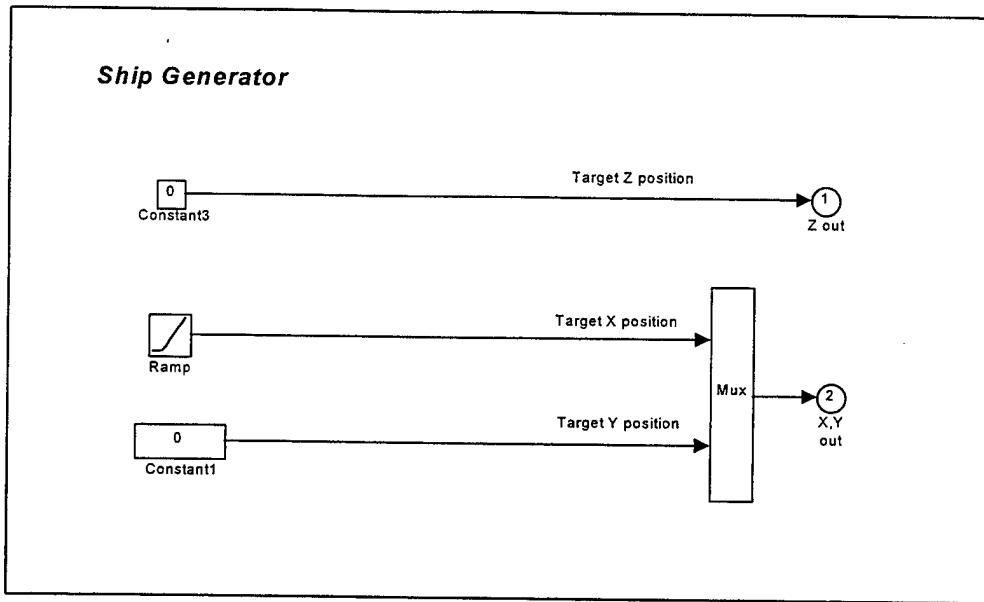


Figure 3.7. Ship Generator Model

In our model the path was set to be 45° from the horizontal plane and the speed was set to 100 ft/sec. The Nulka is launched perpendicular to the ship port side since the model's missile threats always approach by the port side. In our ASCM Digital model, the Nulka decoy always seduces the threat seeker.

1. Description

The Nulka Generator block shown in Figure 3.3 basically has as inputs the selected target (ship) position from the Target Generator Block, the target velocity, the Nulka speed which is assumed to be 100 ft/sec in this model, the Simulink[©] from *FROM* called *flag* (internal parameter, function of *ttg*) and the time of simulation. The target velocity shown in Figure 3.3 has a constant value of 20 ft/sec that must be changed directly in the model for the value of 16.67 ft/sec if the Ship is the target selected. The time-to-go is the remaining time before the threat hits or crosses over the target depending on the type of simulation selected, and basically it defines how much time the

target has to launch the Nulka decoy to seduce the threat seeker. All those inputs are composed in a vector and introduced as inputs for the function *nulka.m* in Appendix D. This program calculates the Nulka position from the beginning of the simulation. It also calculates the exact moment of the Nulka launch as a function of the time-to-go and, after the launch, keeps it in a position relative to the selected target until the end of the simulation.

The *nulka.m* code in Appendix D calculates the Nulka position. After calculating the position, the *nulka.m* code sends that information back to the Nulka Generator block where it is decomposed into the X, Y, Z coordinates and introduced into the Nulka Enabler block shown in Figure 3.9. The Nulka Enabler block sends the Nulka position (or not) to the General Target model depending on the value of the variable *nlk*.

2. *Nulka.m* code

The *nulka.m* code was created to provide the Nulka position calculation from the beginning of the simulation. Initially, the Nulka position coincides with the selected target (ship) position. The launching instant is dependent on the value of *ttg* (time-to-go). When the remaining time before the threat hits or crosses over the target equals the *ttg* value set in the *initial.m* code, the Nulka decoy is launched. The Nulka decoy is launched 45° above the sea surface. Its maximum altitude is set to be 400 feet and the Nulka must keep the relative position in relation to the target constant (port side in our case).

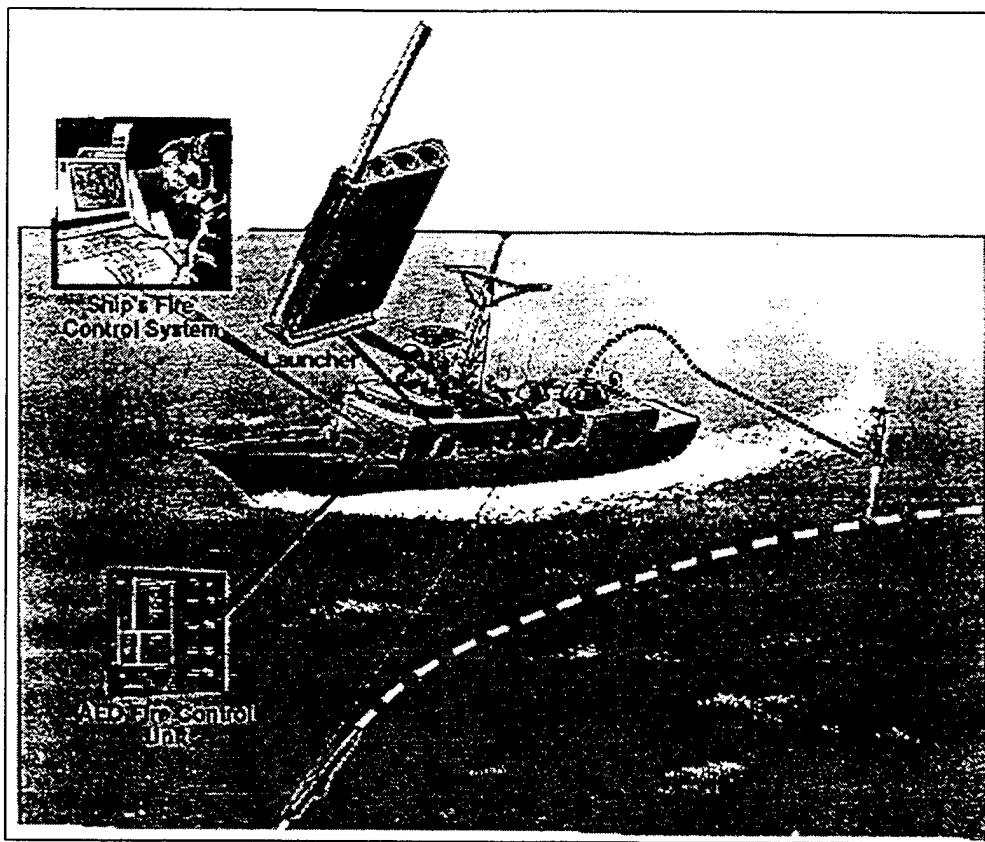


Figure 3.8. Nulka System Components (After [Ref. 8])

The *initial.m* (Appendix B) code starts calling the global variable *po* that was initialized in the *initial.m* code. The *po* variable is composed of five elements: the trigger time, the Nulka X, Y and Z position, and the launching time. In the *initial.m* code, all the *po* elements are equal to zero. When the simulation starts, the current selected target position is sent to the *nulka.m* (Appendix D) function that generates outputs with the same values indicating that, in the beginning of the simulation, the Nulka decoy is onboard. Besides the selected target position, the function also receives the ship velocity, which is 20 ft/sec in this thesis, the Nulka speed (100 ft/sec), the current value of the

internal parameter *flag* sent by the TTG and Miss Distance block (whose initial value is zero, and it changes to one when *ttg* is reached) and, finally, the current simulation time.

During the simulation, the function continuously verifies if the value of *flag* has changed. If the answer is negative, the function output is the current selected target position. If the answer is positive, the trigger time is checked to see if it was already used. If not used, the function re-assigns the current target position and the current simulation time from the second to the fifth elements of *po*. The first *po* element has its value changed to one, which indicates that the trigger time was used. The trigger time ensures that the Nulka decoy is launched only once during a simulation.

For position updating, the function calculates the time difference between the current simulation time and the value stored in *po* (5). The latter stores the simulation time when *ttg* is reached. The time difference, multiplied by the target velocity, generates the increments in the X, Y and Z directions. For the X direction increment, the time difference is multiplied by the target velocity and added to the Nulka initial position stored in *po* (2). For the Y-Z coordinates, the Nulka speed is decomposed in its Y and Z components, multiplied by the time difference and added to the Nulka initial Y (*po* (3)) and Z (*po* (4)) positions. These updated Nulka positions are the function output until the fixed operation altitude (400-ft) is reached. When this altitude is reached, only the X and the Y positions are updated until the end of the simulation.

3. Nulka Enabler

The Nulka Enabler block shown in Figure 3.9 has the function of allowing the presence of the Nulka decoy in the simulation or not. The key variable for this decision is *nlk*. If the *nlk* value is set to one by the *initial.m* code, the simulation has the Nulka

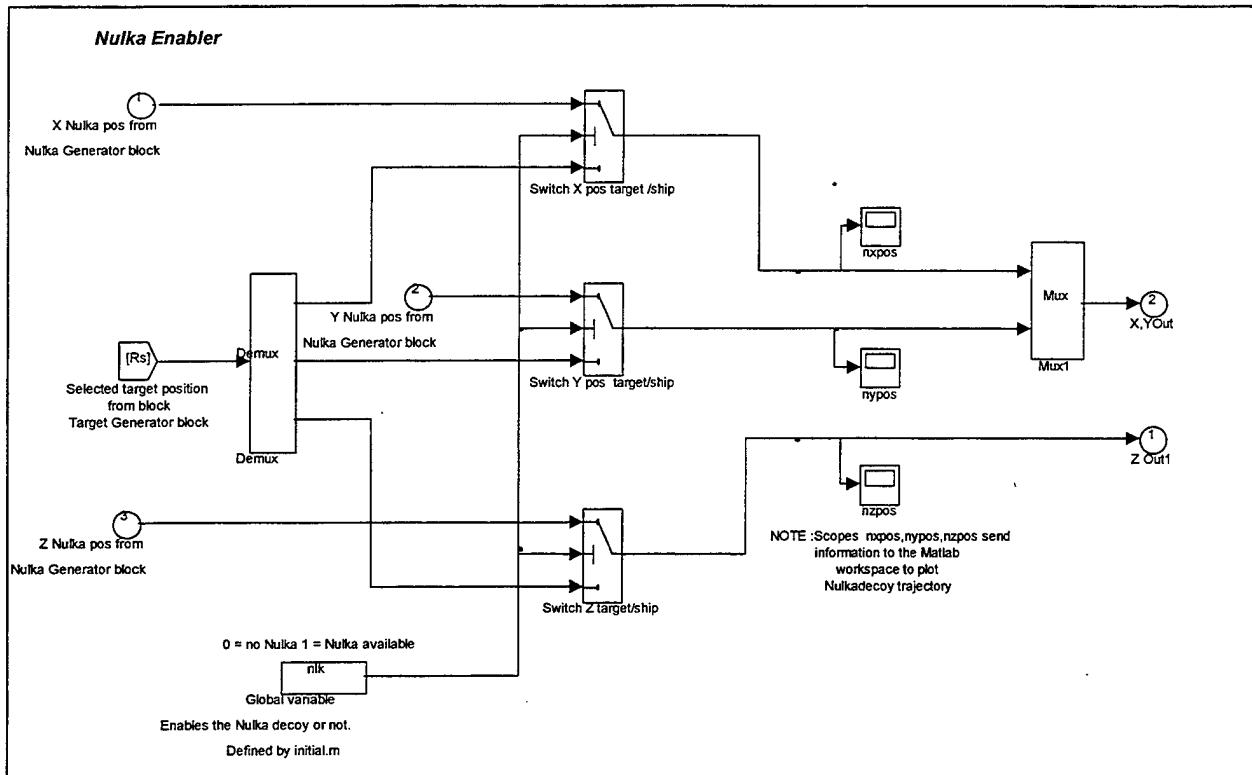
present. On the other hand, if the *nlk* value is equal to zero, the Nulka is not available in the simulation.

The Nulka Enabler receives the Nulka position from the *nulka.m* function as inputs and the selected target position information from the Target Generator block. The Nulka and target X, Y and Z positions are introduced to three switches: one for each coordinate. The threshold value for those switches is unity. The *nlk* value is compared with the threshold values. If *nlk* is one, it releases the Nulka position. If the *nlk* is zero, it releases the target (ship) position. The position released by the three switches is sent to the MATLAB[©] workspace by the Simulink[©] scopes *npos*, *nypos* and *nzpos*. It is evident that if the selected target information is the block output, the general effect on the simulation is no Nulka available.

C. SWITCH TARGET/NULKA

After the target position (calculated by the Target Generator) and the Nulka position (calculated by the Nulka Generator) are obtained, the positions are projected to the Switch Target/Nulka block (Figure 3.10). The basic purpose of this block is to define the exact moment when the Nulka position must start entering in the threat seeker FOV (a function of the *ttg* selected for that simulation).

The internal block structure simply consists of two channels: one for the Z coordinate and the other for the X-Y coordinates. The Nulka and target coordinates are sent from the General Target block and introduced to two switches. The channel switches have unity values as threshold. The value of the internal parameter *flag* is generated by the TTG and MD block, where initially its value is equal to zero. However,



SUMMARY:

BLOCK: Nulka Enabler

SUB-BLOCK(s): None.

INPUT(s): Nulka X, Y and Z position calculated by function *nulka.m*,

Rs, selected target position,

nlk – general switch that allows the presence of the Nulka decoy in the Simulation.

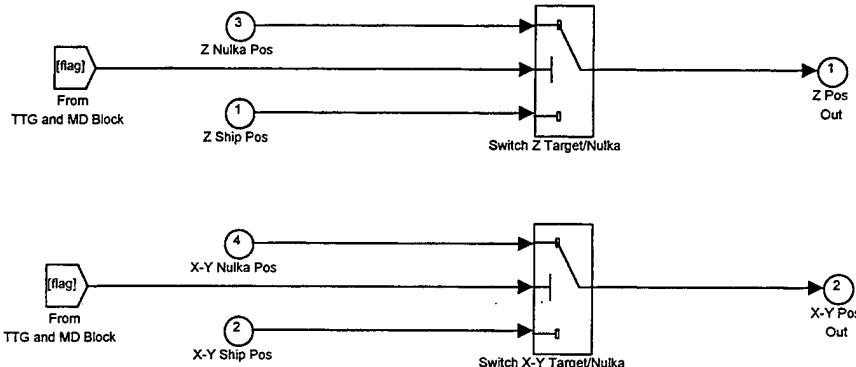
OUTPUT(s): Nulka position

AVAILABLE FUNCTION(s): None.

Figure 3.9. Nulka Enabler Model

when the remaining time before the threat hits or crosses over the target equals the *ttg*, *flag* changes its value to one, which is equal to the threshold value for the channel switches. The Nulka is released as the target position to be offered to the threat seeker in the ASCM Digital model.

Switch Target/Nulka



SUMMARY:

BLOCK: Switch Target / Nulka

SUB-BLOCK(s): None.

INPUT(s): Selected target X, Y, and Z position from General Target Model, Nulka X, Y, and Z position from General Target Model, internal parameter *flag*. Depending on the value of *flag*, it allows the position of the selected target -ship/original target (*flag* =0) or the Nulka position from the Nulka Generator block (*flag* =1) to be projected to the General Target Generator.

OUTPUT(s): General target (Nulka or selected target) X, Y, and Z position to the ASCM Digital Model.

AVAILABLE FUNCTION(s): None.

Figure 3.10. Switch Target/Nulka

IV. INITIALIZATION OF THE MODEL PARAMETERS (INITIAL.M CODE)

To be able to use the ASCM Digital model it is necessary to set initial values for the model parameters. Those variables and their default values are presented in Table 4.1. The MATLAB® *initial.m* code, shown in Appendix B, initializes the default parameters but also executes other important tasks in order to run a large number of simulations.

The default settings of this model represent a specific tactical situation. It is configured to have a target (variable *swtsh* equals to one, Original Target) heading 000° (North) at the speed of 12 knots. The target has a decoy called Nulka (variable *nlk* equals to one) launched as a function of the time-to-go (*ttg*). This target is defending itself from a sea-skimming missile (*carry* equals to one) that has a COSRO seeker (variable *seltrac* equals to one) and a Gaussian beam shape (variable *beam* equal to one). There is no noise present in the seeker (variable *noise* equals zero).

The missile is divided in several sub-systems such as Tracking Seeker, Autopilot and Missile Dynamics. For each sub-system, there are associated variables responsible for the sub-system initializations. For the seeker, there is *ska1* (azimuth forward gain), *ska2* (azimuth feedback gain), *skv1* (vertical forward gain), *skv2* (vertical feedback gain), *sekhlim* (seeker horizontal limit) and *sekvlim* (seeker vertical limit).

Parameter	Default Value	Variable name
Seeker: Azimuth Forward Gain	0.25	<i>skal</i>
Seeker: Azimuth Feedback Gain	0.25	<i>ska2</i>
Seeker: Elevation Forward Gain	0.25	<i>skv1</i>
Seeker: Elevation Feedback Gain	0.25	<i>skv2</i>
Seeker: FOV Horizontal Limit	0.2618	<i>sekhlim</i>
Seeker: FOV Horizontal Limit	0.2618	<i>sekvlim</i>
Seeker :Tracker Type 0=COSRO 1=Monopulse	0	<i>seltrac</i>
Seeker: Beam Shape 0=Gaussian 1=Sinc Function	1	<i>beam</i>
Proportional Navigation: Azimuth	4	<i>enra</i>
Proportional Navigation: Elevation	4	<i>enrb</i>
AutoPilot: Azimuth Forward Gain (Horizontal)	0.5	<i>ta1</i>
AutoPilot: Azimuth Feedback Gain (Vertical)	0.5	<i>ta2</i>
AutoPilot: Azimuth Forward Gain (Horizontal)	0.5	<i>tb1</i>
AutoPilot: Azimuth Feedback Gain (Vertical)	0.5	<i>tb2</i>
Missile Dynamics: XY Acceleration Delay	2	<i>tm</i>
Missile Dynamics: Z Acceleration Delay	2	<i>tmv</i>
Missile Dynamics: Velocity Delay	0.5	<i>tv</i>
Time-to-go	8	<i>ttg</i>
Target Selection 0=Ship 1=Original Target	1	<i>swtsh</i>
Threat Selection 0=Captive-Carry 1=Missile	1	<i>carry</i>
Nulka Enabler 0=No Nulka 1=Nulka on	1	<i>nlk</i>
Noise Enabler 0=No noise 1=Noise present	0	<i>noise</i>

Table 4.1. ASCM Digital Model Default Parameters

The Effective Navigation Ratio is not a sub-system but it introduces a gain to the L.O.S. (line-of-sight) rate from the Tracking Seeker Dynamics block in both the horizontal and vertical channels of the Autopilot. The gain values are set by the variables ***enra*** and ***enrb***. The Autopilot has the variable ***ta1*** (azimuth forward gain), ***ta2*** (azimuth feedback gain), ***tb1*** (vertical forward gain) and ***tb2*** (vertical feedback gain). For the Missile Dynamics the variables responsible for the missile initialization are ***tm*** (XY Acceleration Delay), ***tmv*** (Z Acceleration Delay) and ***tv*** (Velocity Delay). Finally, the remaining two variables are ***ttg*** and ***chaff***. The variable ***chaff*** is not associated with any sub-system or special target countermeasure. It was put in the initialization code and in the Graphical User Interface (GUI) *Menu* to prepare this model for further improvements. The variable ***ttg*** (time-to go) defines the exact moment when the Nulka decoy is launched in order to seduce the threat seeker. This seduction makes the missile break-lock which gives a miss distance or effectiveness measurement.

The *initial.m* code defines initially two global variables, ***po*** and ***I***. The variable ***po*** is a 1x5 matrix responsible for launching the Nulka rocket after the time-to-go set for the simulation being reached. Each element in the matrix has information that will be used for the *nulka.m* code shown in Appendix D and the Nulka Generator block in the ASCM Digital Model.

The global variable ***I*** is a structure responsible for storing all the initialization model parameters. The fields of the structure are presented in Appendix B, but are also introduced here. The initialization parameters were divided into four major groups: General Switches, the ASCM Digital Model, the COSRO Seeker and the monopulse Seeker. The General Switches group is responsible for setting the tactical environment

where the simulation is to happen, i.e., the number of the simulation (variable ***num*** indexes all the other parameters), if the target is the Original Target or Ship (variable ***swtsh***), if the threat is a missile or a captive-carry (variable ***carry***), if the Nulka decoy is enabled or not (variable ***nlk***), if the threat seeker has noise in it or not (variable ***noise***), and what type of seeker is available for the simulation (variable ***seltrac***). The variables ***ska1***, ***ska2***, ***skv1***, ***skv2***, ***enra***, ***enrb***, ***ta1***, ***ta2***, ***tb1***, ***tb2***, ***tm***, ***tmv***, ***tv***, ***ttg***, ***sekhl*** and ***sekvl*** compose the second group or the ASCM Digital Model. These variables set up the initial values of the ASCM Model.

The seeker initialization parameters are very similar in their purpose as they set the values for the type of seeker selected by ***seltrac*** (COSRO or monopulse). The parameters for the COSRO seekers are: ***ppc*** (peak power, in kW), ***freqc*** (frequency, in GHz), ***antgainc*** (antenna gain, in dB), ***HPc*** (half power beamwidth, degrees), ***noibwc*** (noise bandwidth, in MHz), ***noifigc*** (noise figure, in dB), ***rrc*** (Range resolution, in meters), ***numpulc*** (number of pulses integrated), ***nosangc*** (normalized offset angle), ***envc*** (envelope correlation time, in seconds), ***nutfreqc*** (nutation frequency, in Hertz), ***serbwc*** (servo bandwidth, in Hertz), ***crossc*** (target radar cross subsection, in square meters), ***alphac*** (one-way atmospheric attenuation, in dB/km) and ***beam*** (which selects the beam shape for the COSRO seeker, Gaussian or Sinc function).

The parameters of the monopulse seeker are: ***ppm*** (peak power, in kW), ***freqm*** (frequency, in GHz), ***antgainm*** (antenna gain, in dB), ***HPm*** (half power beamwidth, degrees), ***noibwm*** (noise bandwidth, in MHz), ***noifigm*** (noise figure, in dB), ***rrm*** (Range resolution, in meters), ***numpulm*** (number of pulses integrated), ***fslrm*** (first side lobe ratio, in dB), ***crossm*** (target radar cross subsection, in square meters) and ***alpham*** (one-

way atmospheric attenuation, in dB/km). These seeker parameters are fundamental to the noise calculation by the *noise_error.m* code shown in Appendix C. After defining *po* and *I*, the *initial.m* code creates 49 cells to store the initialization values.

The next step is to assign the initial values for the number of simulations required. In this thesis, for instance, the number of simulations was set to 4,800 due to the missile performance analysis when changing the ASCM Digital Model parameters. However, that number depends on the purpose to be attained. The default values in this thesis for the ASCM Digital Model parameters are presented in Table 4.2 and may vary. The COSRO and monopulse initial values used for the noise calculation remain the same as in Chapter II.

General Switches values in the *initial.m* code are presented in Appendix B with the following assumptions: the values of *num* range from 1 to 2,400; all the simulations set by *initial.m* code are for the missile as a threat (*carry* equals to one) to the Original Target (*swtsh* equals to one). The Original Target always has the Nulka decoy enabled (*nlk* equals to one).

For noise calculation, there are four distinct groups of simulations depending on each variation of the parameters presented in Table 4.2. The first group of simulations assumes there is no noise. The second group assumes that there is noise and it affects a COSRO seeker with a Gaussian beam shape. The third group also assumes that there is noise and it affects a COSRO seeker with a sinc function beam shape. The fourth and last group also assumes there is noise and that it affects a Monopulse seeker. The COSRO and monopulse seeker parameters do not change during the simulations run in Chapter II.

Parameters	Range of Values
<i>ska1,ska2,skv1 and skv2</i>	[0.1 ,0.25 (d)*, 1 , 4]
<i>enra and enrб</i>	[1 , 2 , 4 (d) , 8]
<i>ta1,ta2,tb1,tb2 and tv</i>	[0.25 , 0.5 (d) , 1 , 4]
<i>tm and tmv</i>	[0.5 , 1 , 2 (d) , 4]
<i>sekhlim and sekvlim</i>	[0.1309 , 0.2618 (d) , 0.3491 , 0.5236]
<i>ttg</i>	[2 , 4 , 6 , 8 (d) , 10 , 12 , 14 , 16 , 18 , 20]
* (d) = default value	

Table 4.2. ASCM Digital Model Initial Values Range

The *I* structure is formed by assigning the cells with the simulation data to the fields created to store them. The fields have the name of each variable presented before. After the fields receive the simulation data, the Graphical User Interface (GUI) *Menu* shown in Appendix E is called up and allows a user to select the simulation number and its associated variable values to initialize the ASCM Digital Model and to set the engagement conditions.

The following step is to assign the model default values that are in the first position of the structure *I*. These default values are associated with the first simulation and are presented in Table 4.2.

Finally, a vector called *current* is defined that is used as input data to the Noise Generator block in the ASCM Digital Model. The *current* vector introduces the default values of the COSRO and Monopulse seeker parameters for the noise calculation using the function *noise_error.m* shown in Appendix C.

V. OPEN-LOOP CAPTIVE-CARRY MODEL

A. THE CAPTIVE-CARRY EXPERIMENTS

The purpose of this chapter is to discuss the open-loop captive-carry configuration of the model and compare the results with the closed-loop configuration. One example of a captive-carry aircraft used today by the Naval Research Lab (NRL) is a modified P-3 shown in Figure 5.1.

One of the major differences between the captive-carry configuration and the actual missile configuration is its speed. In our simulations, the missile speed is 1322 ft/sec while the captive-carry speed is 400 ft/sec. The speed differences generate two distinct time frames for an analysis of the results.

One other difference is that while the actual missile flies toward the target using the guidance signals from its seeker, the seeker signals in the captive-carry experiments are totally ignored by the captive-carry aircraft dynamics. Thus, it is possible to use the actual missile configuration to get the electronic attack (EA) effectiveness (miss distance, in our case) but it is more difficult for the captive-carry configuration. The results obtained from the captive-carry model configuration are the seeker range-to-target and the antenna azimuth and elevation angles.

B. CAPTIVE-CARRY MODEL

1. Captive-Carry Dynamics

The Captive-Carry Dynamics block is shown in Figure 5.2. This is part of the Missile/Captive-Carry Dynamics block shown in Figure 2.2 and generates

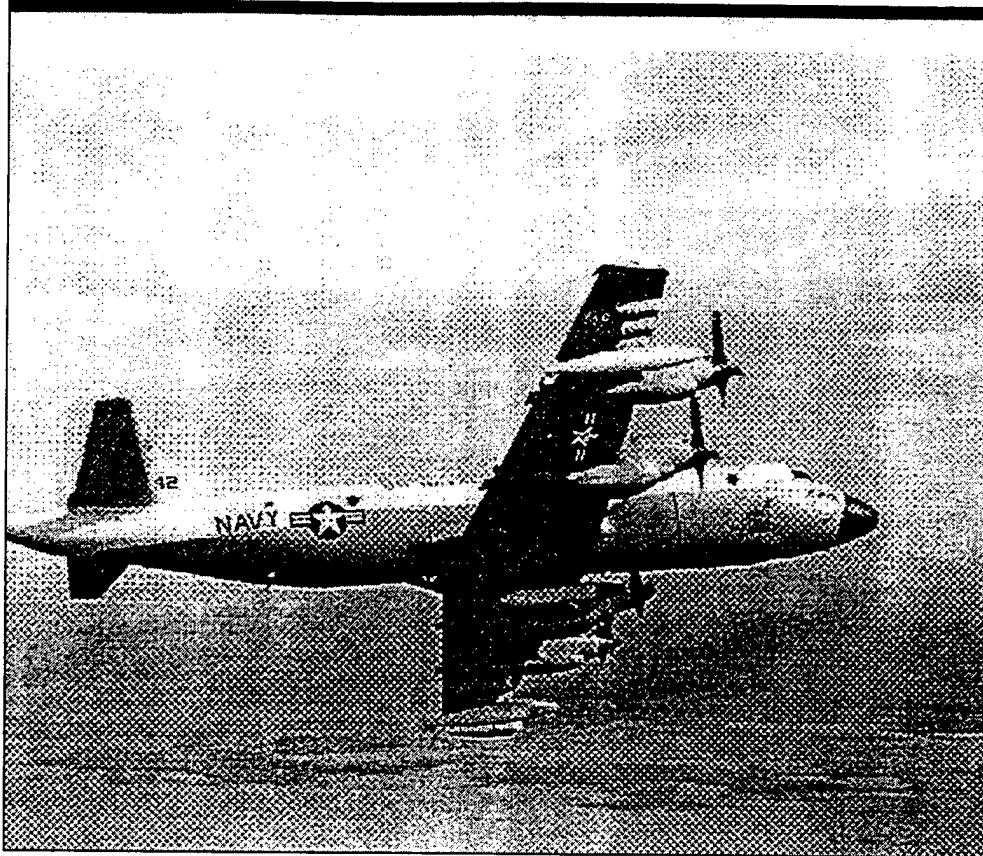


Figure 5.1. NRL P-3 Captive-Carry Research Aircraft Carrying the HIL Simulators

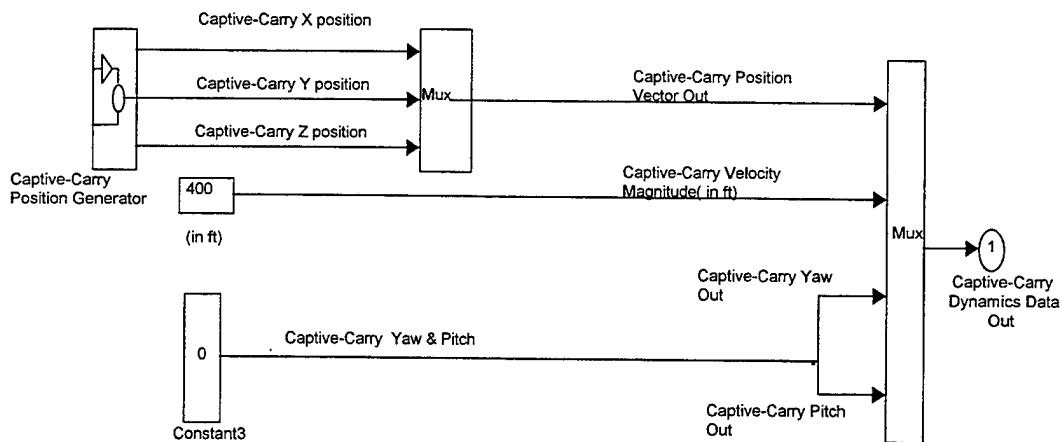
the aircraft dynamics data (position, velocity magnitude, pitch and yaw) required to simulate an aircraft flying as shown in Figure 5.3. It is composed of the captive-carry aircraft constantly adjusts its flight profile in order to cross over the ship. The Captive-Carry Position Generator sub-block shown in Figure 5.4 and two constant Simulink[©] sources that generate the captive-carry velocity (400 ft/sec), and the pitch and yaw information. For this model it was assumed that the pitch and yaw angles of the aircraft have values equal to zero in order to better identify the difference in seeker response between the two configurations.

In the Captive-Carry Position Generator block, the aircraft altitude is kept constant at 1000 feet during the simulation. For the X-Y plane, during each instant of the simulation, the aircraft to the current target position and the simulation is stopped when the aircraft crosses over the target as shown in Figure 5.3.

To implement these requests shown in Figures 5.4 and 5.5, the block receives the target X-Y position from the Target Generator block that is subtracted from the current aircraft position and thus generates a relative increment in the X and Y directions. Those increments are combined by a multiplexer block and introduced in a MATLAB[©] *function* block, which calculates the angle between them. The angle is used to calculate the aircraft X and Y current position by multiplying the cosine and sine of the angle by the aircraft speed.

The aircraft X-Y positions are sent together with the aircraft altitude to the Captive-Carry Dynamics block where the position information is combined in a 3 x 1 vector format and, and later combined again in a larger vector format with the other dynamic data generated in this block.

Captive-Carry Dynamics



SUMMARY:

BLOCK: Captive-Carry Dynamics

SUB-BLOCK (s): Captive-Carry Position Generator

INPUT(s): None.

OUTPUT(s): Captive-Carry Dynamics Data:
Position,
Velocity Magnitude,
Yaw and
Pitch

AVAILABLE FUNCTION(s): None

Figure 5.2. Captive-Carry Dynamics Block

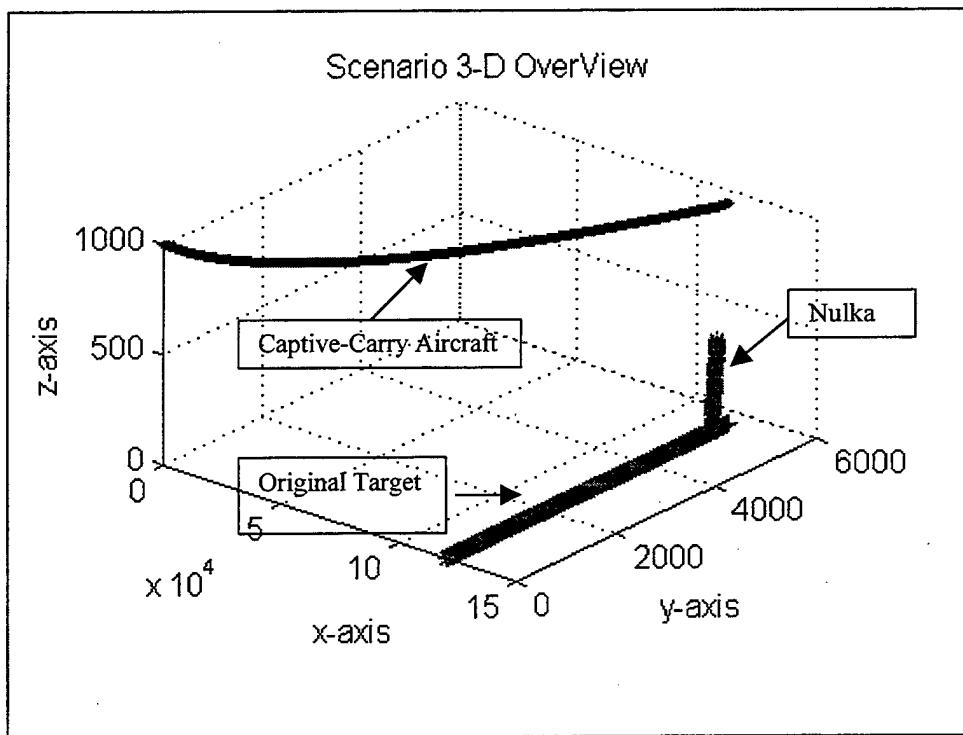
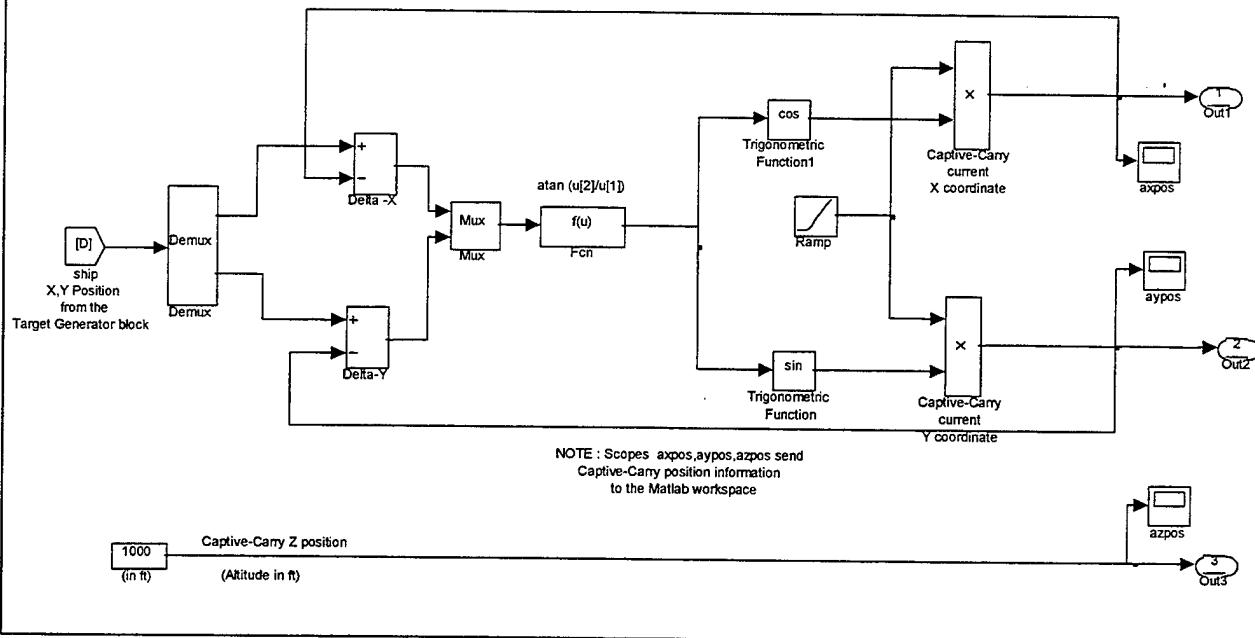


Figure 5.3. Captive-Carry Trajectory

Captive-Carry Position Generator



SUMMARY:

BLOCK: Captive-Carry Position Generator

SUB-BLOCK(s): None.

INPUT(s): Ship X-Y Position from Target Generator block

OUTPUT(s): Captive-Carry Position

AVAILABLE FUNCTION(s): None.

Figure 5.4. Captive-Carry Position Generator Block

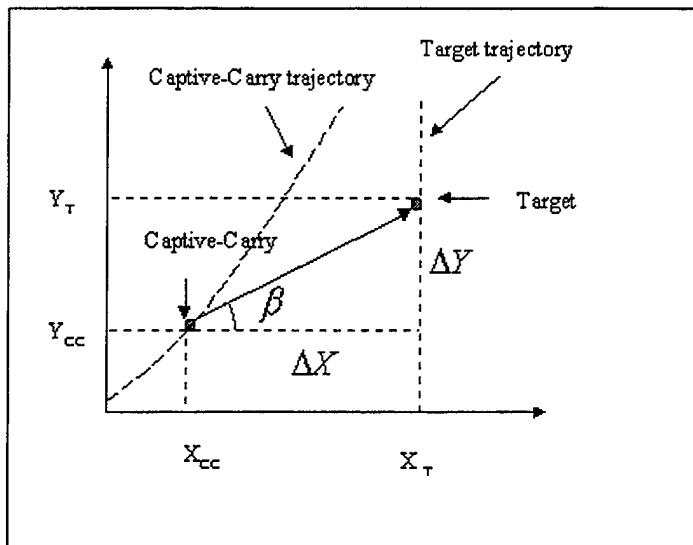


Figure 5.5. Example of Captive-Carry and Target Trajectories

Finally, when the aircraft crosses over the target, a singularity is calculated due to the target and aircraft position coincidence in the X or Y dimension and the simulation is stopped.

2. Selection of the Captive-Carry Experiment

For this thesis, both the closed and open loop simulations are available to be run using the GUI (Graphical User Interface) *Menu* (Appendix E). The next chapter shows how to select between the open and closed-loop configurations.

VI. GRAPHICAL USER INTERFACE (GUI) IMPLEMENTATION FOR ASCM DIGITAL CONTROL

A. GUIDELINES FOR BUILDING A GUI

A large number of simulations are needed in order obtain get data for studying the effects of the ASCM Digital Model parameters on the seeker responses. This large amount of data was the determining factor in designing a Graphical User Interface (GUI). The GUIs create a better user work environment, easily handle the initialization model as well as stores all the information generated by the simulation.

The design of the GUI followed a few basic principles: simplicity, harmony and friendliness. The goal of simplicity was to include in the GUI only the information that would be useful for the user such as what general switches were selected by clicking the simulation number on the popup list, updating the ASCM and seeker parameters automatically, and offering push buttons to run, store and graph the missile and captive-carry simulations. Harmony was reached by creating a few buttons to perform several functions without user supervision and confusion. Harmony makes it extremely easy to run the missile and captive-carry simulations without any big changes in the basic procedure described below.

Friendliness is aimed at allowing a user to learn very quickly how to run, save, store, and graph all simulation information for further analysis without being knowledgeable about the ASCM model. All the GUIs in this thesis and those presented

in this section were created using the MATLAB[©] tool called *Guide* that allows a friendly user interface to create a GUI.

B. GRAPHICAL USER INTERFACES FOR A LARGE NUMBER OF SIMULATIONS

Two GUIs exist for running the large number of simulations. One GUI is for running simulations and data storing and the other is for graphing the simulations for both missile and captive-carry experiments. The GUI for running the simulations is called *Menu* and is presented in Figure 6.1. Its associated source code, *menu.m*, is in Appendix E. The GUI for graphing the simulations is called *simulations_plot* and it is presented in Figure 6.2. Its MATLAB[©] code, *simulations_plot.m*, is presented in Appendix F.

First, the GUI *Menu* is described. In Figure 6.1, eight work areas are divided by the following frames: Select Missile Simulation, Basic Settings, frame for running missile simulation, frame for captive-carry simulation, frame for graphing and exiting the GUI and also storing the simulation data in the MATLAB[©] workspace, ASCM Model Parameters, COSRO Parameters and Monopulse Parameters. The Select Missile Simulation frame has a popup list with odd numbers representing the closed-loop missile simulations and their parameters associated by the *initial.m* code and the structure *I*. The default values are associated with the first simulation. The *update.m* code shown in Appendix G is the function associated with this popup list.

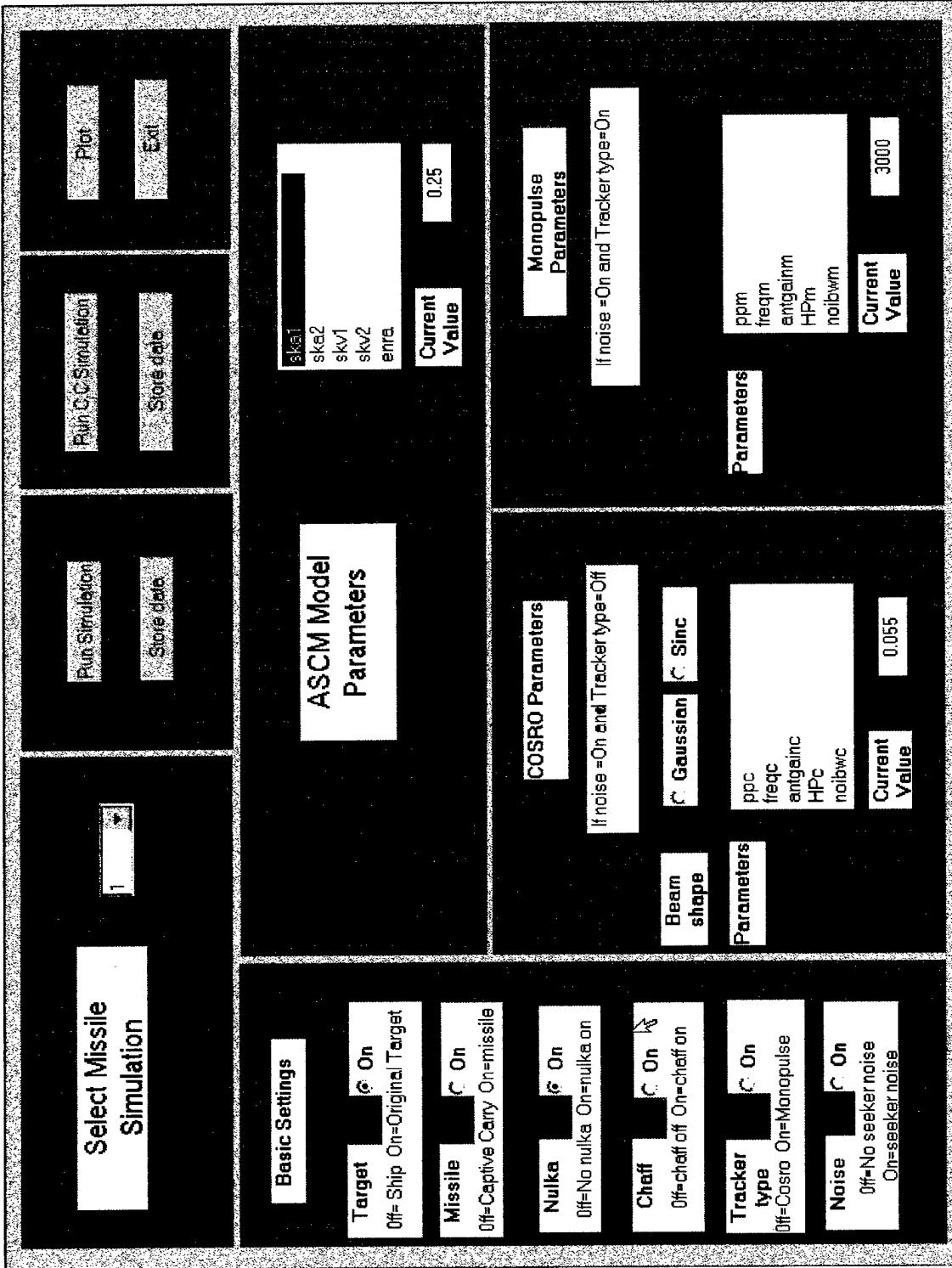
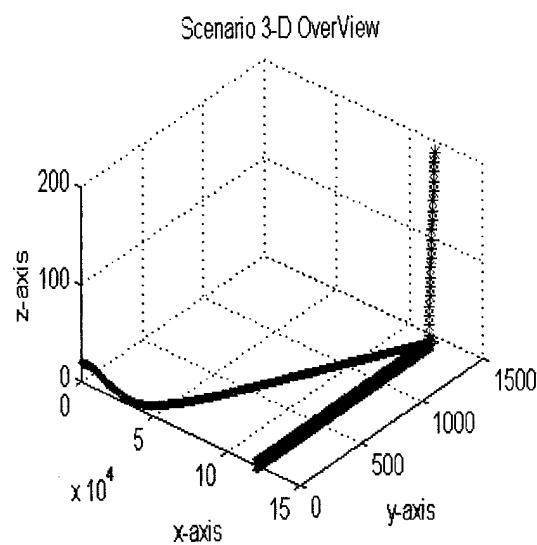


Figure 6.1. GUI Menu

The Basic Settings is a model setting status frame where the current selections of the model general switches are displayed depending on the simulation number selected. The Basic Settings frame shows if the simulation has the original target (“On”), target heading 000° at speed 12 knots, or has a ship (“Off”) as target heading 090° at 10 knots. It also shows the type of simulation selected, closed-loop (“On”) or the open-loop (“Off”); if the EA (Electronic Attack) Nulka decoy is available on board (“On”) or not (“Off”); if the tracker type for the simulation will be a COSRO, default, (“Off”) or a monopulse seeker (“On”), because the value of noise is a function of which seeker is selected and finally, if there will be a presence of noise (“On”) or not (“Off”) in the seeker of the model. There is also a field called Chaff, but in the present model, this function is not available. It was left on the GUI for further improvements in the model and to aid in the development of special features required later.

When the Tracker Type field is selected, COSRO (“Off”), it enables the radio buttons in the COSRO Parameters area. These parameters are related to the Beam shape (Gaussian - default- and *sinc* function). If the Monopulse seeker is selected (“On”) both radio buttons will be turned off. The frame running missile simulations has two buttons: the *Run Simulation* button (up) and the *Store data* button (down). The first one only calls up the ASCM model and allows a user to run the selected simulation from the Simulink© ASCM Digital Model block. The latter is associated with the function *store_simulation.m* shown in Appendix I.



Simulation Graphs

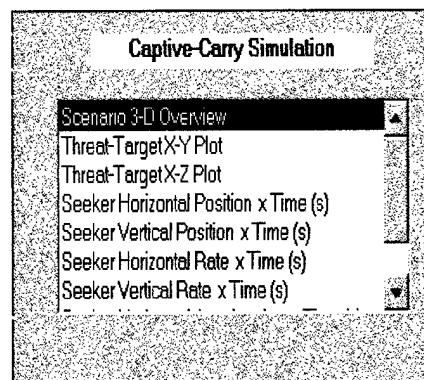
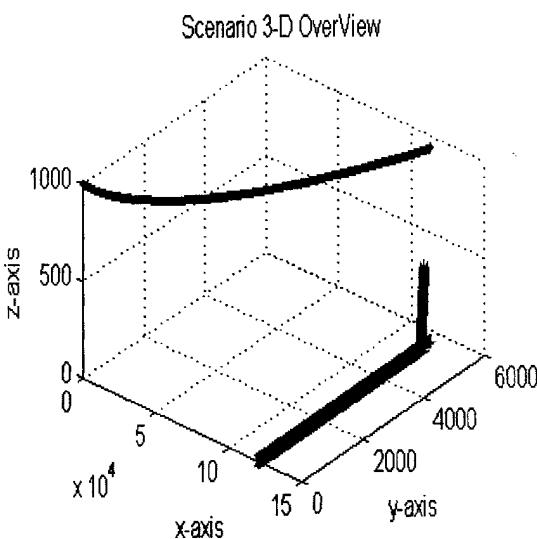
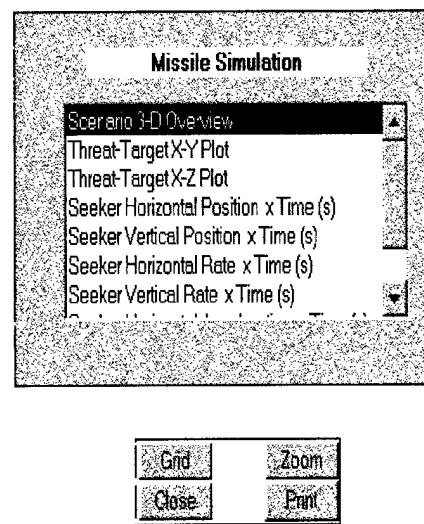


Figure 6.2. GUI Simulations_plot

The field for running captive-carry simulations also has two buttons: *Run C.C Simulation* (up) and *Store data* buttons (down). The *Run C.C simulation* runs the *update1.m* function shown in Appendix H and also calls up the ASCM model. The *Store data button* uses the function *store_simulation1.m* shown in Appendix J.

The next area is the one related to graphing the data stored for the selected simulation in the Select Missile Simulation list. The button Plot calls up the GUI *simulations_plot* shown in Figure 6.2. The other button in this area has a function to store the MATLAB© workspace containing the *store* structure which stores all the simulation data in a matrix called *gold* and also to close the GUI *Menu*.

The remaining frames, ASCM Model, COSRO and Monopulse Parameters, have the same format. First, they have a popup list with the parameters associated with the ASCM Digital model, with the COSRO seeker and with the Monopulse seeker, respectively. By scrolling and clicking on a specific parameter, the associated value of the selected parameter indexed by the simulation number in the structure *I* will be updated by *update.m* shown in Appendix G and *update1.m* shown in Appendix H and displayed below the popup list beside the *Current Value* text.

The purpose of the GUI *simulations_plot* after running a pair of simulations is to graph the information stored by the functions *store_simulation.m* shown in Appendix I for the closed-loop simulation, and *store_simulation1.m* shown in Appendix J for the open-loop simulation.

On the left side of the GUI in Figure 6.2, there are two graphs related to the missile (top figure) and captive-carry (bottom figure) simulations. On the right side, two

large frames called *Missile Simulation* and *Captive-Carry Simulation* enclose the six possible selections of graphs available: Scenario 3-D, Threat-Target X-Y Plot, Threat-Target X-Z Plot, Seeker Horizontal Position x Time, Seeker Vertical Position x Time, Seeker Horizontal Rate x Time, Seeker Vertical Rate x Time, Seeker Horizontal Acceleration x Time, and Seeker Vertical Acceleration x Time.

The Scenario 3-D plot is the tri-dimensional representation of the players involved in the simulation: the threat (missile and captive-carry), the selected target (original target or ship) and Nulka trajectories. The Threat-Target X-Y and X-Z Plot graphs are representations of the same scenario of threat-target-Nulka but only in two dimensions. The other graphs, involving the seeker performance (position, rates and accelerations), are available to allow a user to measure, compare and study the seeker responses during the closed and open-loop engagement. Lastly, there is a small frame for both graphs that makes it possible to set the grid on the graphs (2-D and 3-D graphs), to zoom in on 2-D graphs and to print the present screen or close the GUI.

C. GUI CODES

1. *Update.m* Code

This code, shown in Appendix G, is associated with the popup list on the *Select Missile Simulation* frame. *Update.m* gets the value of the string index where the simulation number in the popup list is stored and converts it into two index values. One index called *num* is used to access the structure *I* and get the values of the model parameters. The other index is called *index* and is calculated to be an odd value to store the missile simulation results in the *store* structure. For executing several sequential

simulations the first element of the global variable *po* (trigger time) is reset each time that *update.m* is called up. Also, each time *update.m* is called up it sets the values of the status radio buttons in the Basic Settings area associated with the value of *num* for the selected simulation. The General Switches are updated for a selected simulation as are the COSRO and monopulse parameters with the values indexed in the *I* structure by the index *num*. Finally, the values of the vector *current* (COSRO and Monopulse parameters) used for noise calculation are updated with the values of the selected simulation.

2. *Update1.m* Code

The *update1.m* code, shown in Appendix H, is associated with the *Run C.C Simulation* pushbutton in the captive-carry simulation frame. First, *update1.m* reinitializes the Nulka trigger time and then it changes the value of the storage *index* to get the even valued simulations. The even valued positions are used to store the captive-carry results in the structure *store*. The value of the variable *carry* in the initialization structure is set to zero. Finally, the threat status radio button in the Basic Settings is updated to reflect the captive-carry simulation. The other parameters are the same as those of the closed-loop simulation where the parameters are indexed by *num* because similar conditions are required for both experiments except the threat vector and the parameters cited above. The values of the vector *current* (COSRO and monopulse parameters) used for noise calculation in captive-carry simulation are also updated with the values of the selected simulation.

3. *Store_simulation.m* Code

The *store_simulation.m* code is shown in Appendix I is activated by pressing the *Store data* pushbutton in the missile simulation area. It creates cells for storing the miss distance (after its calculation), simulation time, seeker performance data (position, rate and acceleration) for the azimuth and vertical channels, the ASCM model parameters, the COSRO/Monopulse seeker parameters, and the missile, target and Nulka position. These values are stored in the *store* structure (odd positions store missile simulations, and even positions store captive-carry simulations).

4. *Store_simulation1.m* Code

This code shown in Appendix J is associated with the *Store data* pushbutton in the captive-carry simulation area. It creates cells for storing miss distance (typically a constant value in the captive-carry experiment), simulation time, seeker performance data, the ASCM model parameters, the COSRO/monopulse parameters and the captive-carry, ship and Nulka position. The last step executed by this function is to set the *carry* variable value back to one. The Nulka trigger time for the next simulation is reinitialized and decreases by one the storage *index*.

5. *Plot1.m* and *Plot2.m* Codes

The *plot1.m* and *plot2.m* codes are described together because there are no major differences between them except for the *index* value that accesses the values stored in the structure *store*. For odd index values (*plot1.m*), the missile simulations data are stored in auxiliary variables and, after that, depending on the graph selection made in the Missile Simulation frame, the graphs are presented in the upper left side on the GUI

simulations_plot. For even *index* values, the procedure is the same, except that the captive-carry position and the results associated with the open-loop simulations are the ones stored and graphed in the left bottom side of the GUI.

D. PROCEDURES TO RUN SIMULATIONS USING MENU

A few steps are necessary to run, store and graph simulations involving both the missile and captive-carry threat for the same tactical conditions. They are briefly discussed in this section.

The first step in the MATLAB© prompt is to call up the *initial.m* code and set the initialization model parameters with their default values. The GUI *Menu* will pop up after the initialization process.

The second step is to select the simulation number desired. The *update.m* code automatically updates the Basic Settings status radio buttons as well as the other model parameters for the missile simulation.

The third step is to press the *Run Simulation* pushbutton in the missile simulation area that calls up the Simulink© ASCM Digital Model.

The fourth step stores the missile simulation results by pressing the *Store data* pushbutton in the same missile simulation frame calling up the *store_simulation* code.

The fifth step is to switch to the captive-carry simulation by pressing the *Run C.C Simulation* in the captive-carry simulation area and calling up the *update1.m* code.

The sixth step is to store the captive-carry simulation data.

If simulation graphs are desired pushbutton *Plot* calls up the GUI *simulations_plot*. If more simulation runs are required, the user must start over from Step

1 again. It is also possible to run all the simulations in the Select Missile Simulation number list, to store all of them and then to compare the simulation results in the *simulations_plot* GUI for the selected simulation number. The last procedure was the one used for this thesis due to the large number of simulations required.

VII. COMPARISON OF SEEKER SIGNALS

The results of the captive-carry simulations are the seeker angles and the range-to-target. The closed-loop simulations, on the other hand, give us the additional information of a miss distance.

Despite the differences between the closed-loop and open-loop simulations, it is instructive to compare the seeker responses from both configurations. Several simulations (simulations 313, 633, 953, 1273, 1593, 1913, 2233, 2553, 2873, 3193, 3513, 3833, and 4153 in Appendix M) were run to give the data necessary to compare the seeker signals of each simulation.

The simulations used to get, to graph and to compare the seeker output results were obtained utilizing the following settings:

- The Original Target is the target selected (*swtsh* =1);
- The Nulka decoy is activated (*nlk* =1);
- The threat seeker is monopulse (*seltrac* =1);
- There is noise in the monopulse seeker (*noise* =1); and
- The time-to-go setting is 14 seconds (*ttg* =14).

The model parameters, *ska1*, *ska2*, *skv1*, *skv2*, *enra*, *enrb*, *ta1*, *ta2*, *tb1*, *tb2*, *tm*, *tmv*, and *tv*, were set to their highest values in Table 4.2. The resultant graphs present the missile and the captive-carry seeker responses for each parameter mentioned above. The seeker responses for this model are the seeker horizontal position, seeker vertical

position, seeker horizontal rate, seeker vertical rate, seeker horizontal acceleration and seeker vertical acceleration.

After comparing the results from the graphs an algorithm can be developed in order to obtain an accurate miss distance prediction from the captive-carry results. An example of this type of an algorithm is discussed in [Ref. 9].

In this thesis, a similar approach is developed. This approach consists of using a neural network to predict a missile trajectory from the captive-carry experiment in order to get the effectiveness measurement prediction. The neural network approach is made possible because of the range of values of the closed-loop experiments used to train the neural network are similar to the results obtained from the captive-carry experiment, as is shown in the following graphs.

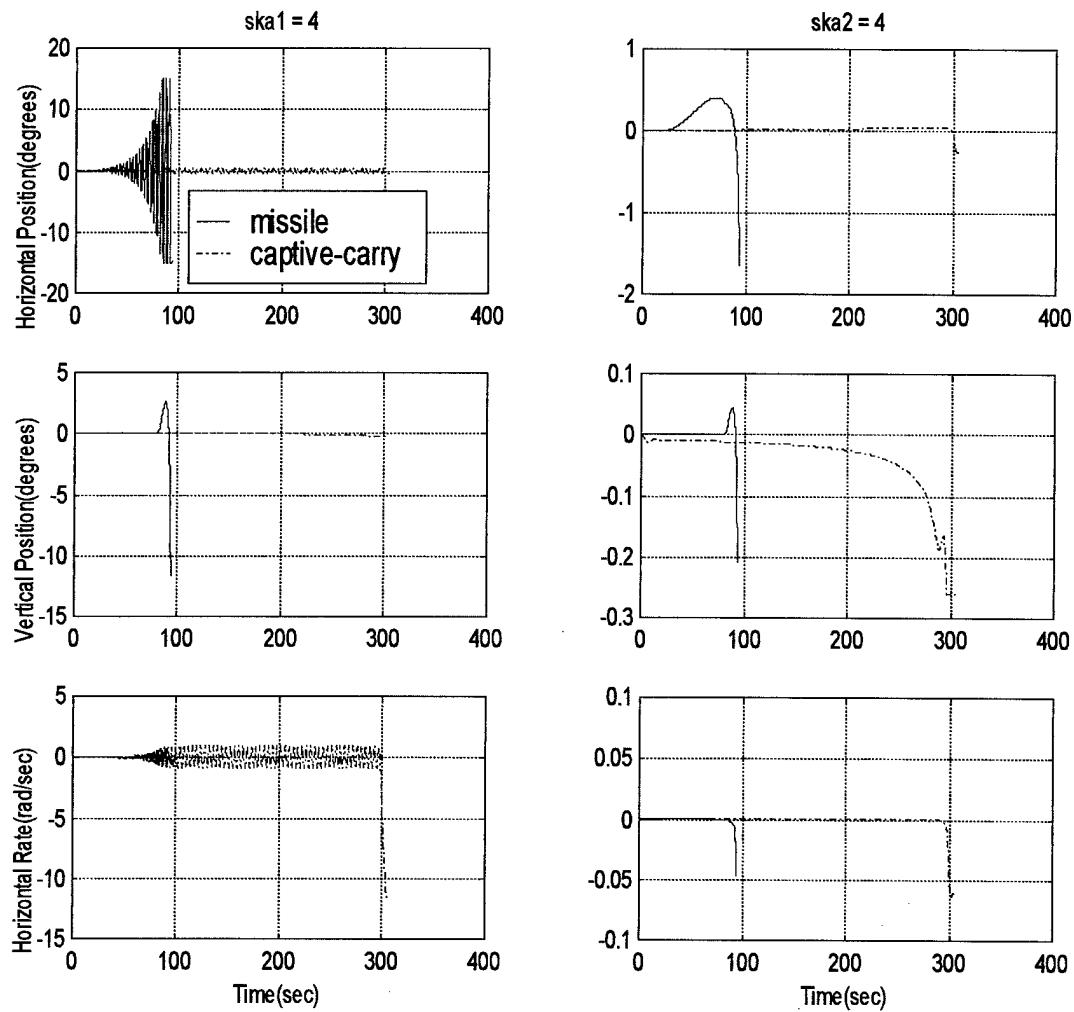


Figure 7.1. Seeker Response Comparison for *ska1* and *ska2*

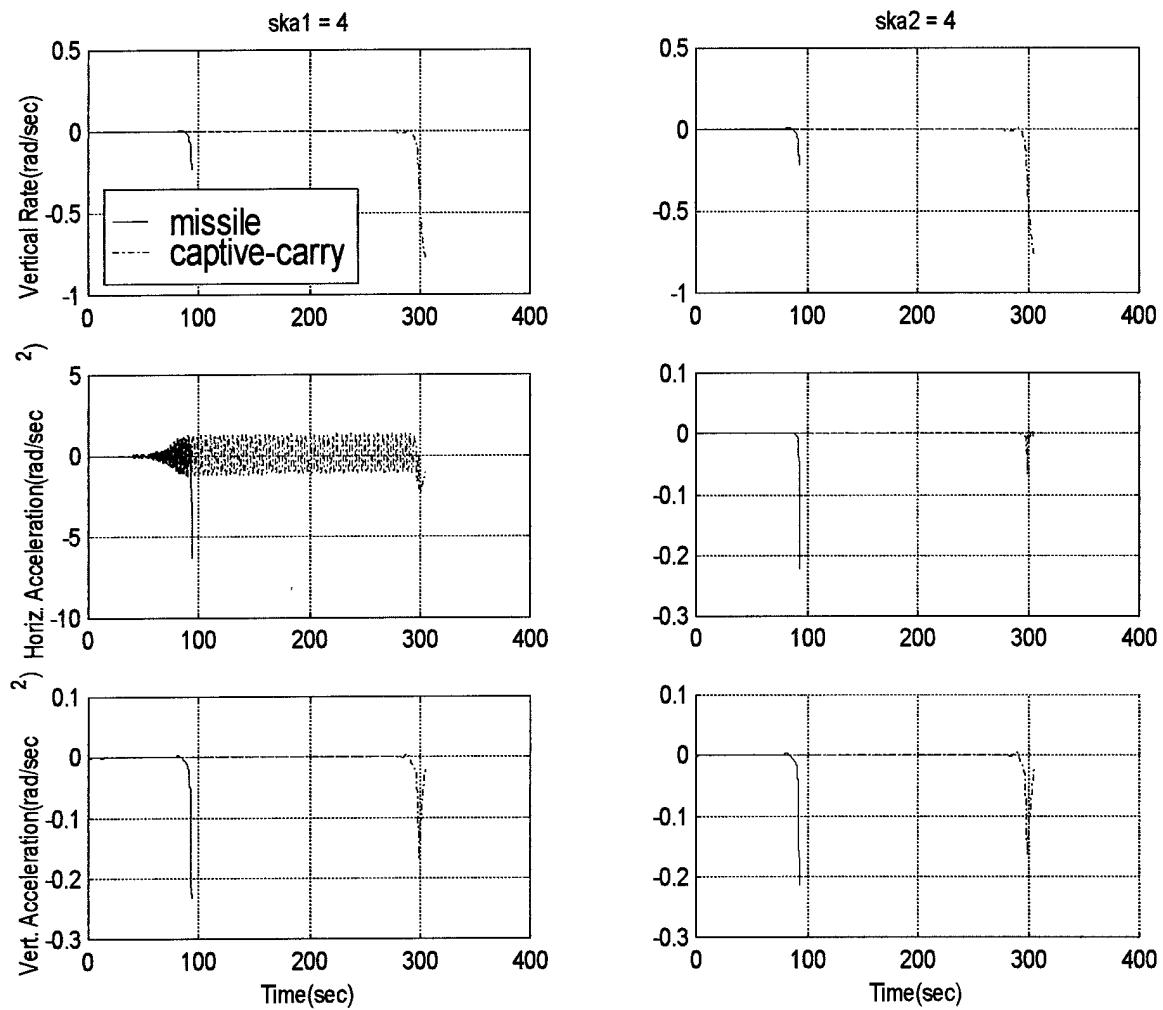


Figure 7.2. Seeker Response Comparison for $ska1$ and $ska2$

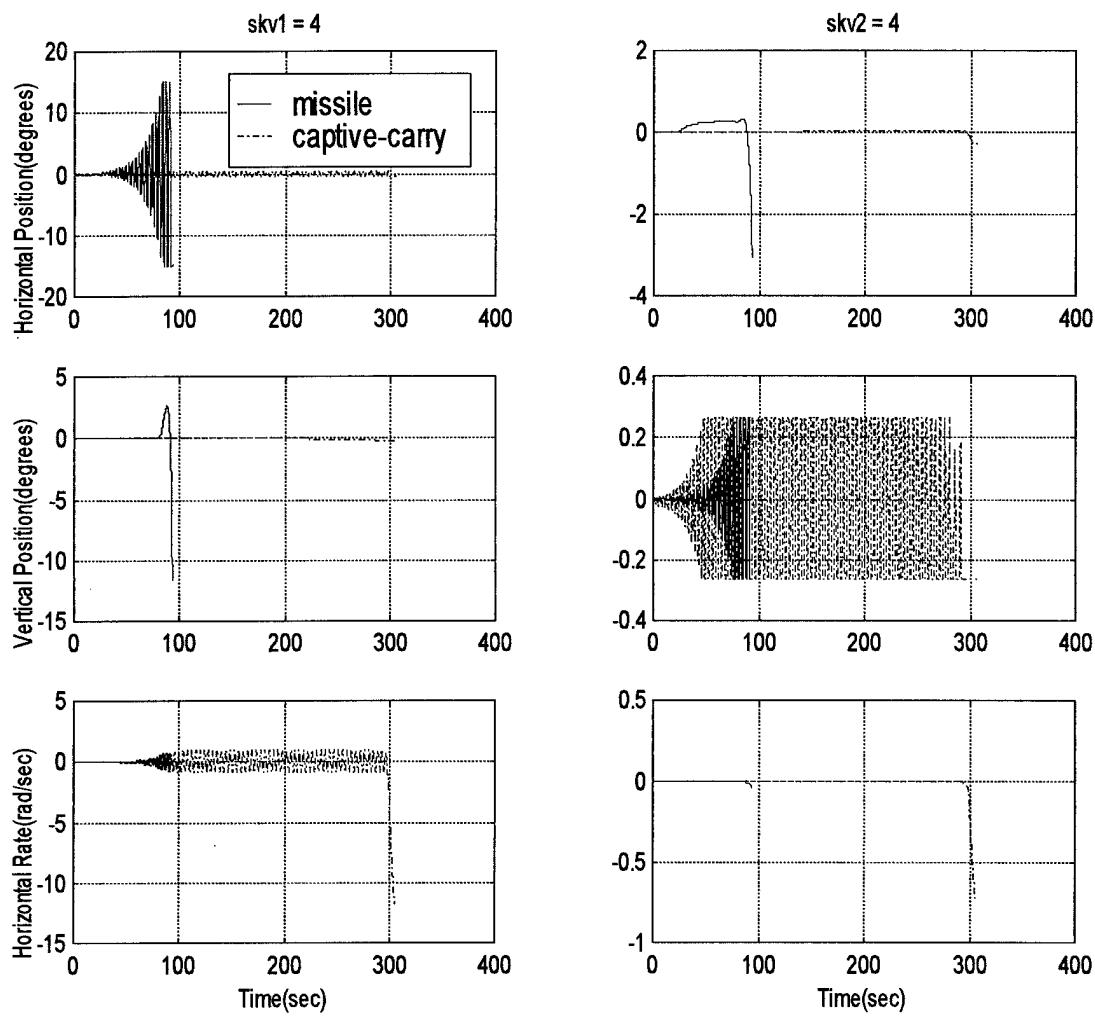


Figure 7.3. Seeker Response Comparison for $skv1$ and $skv2$

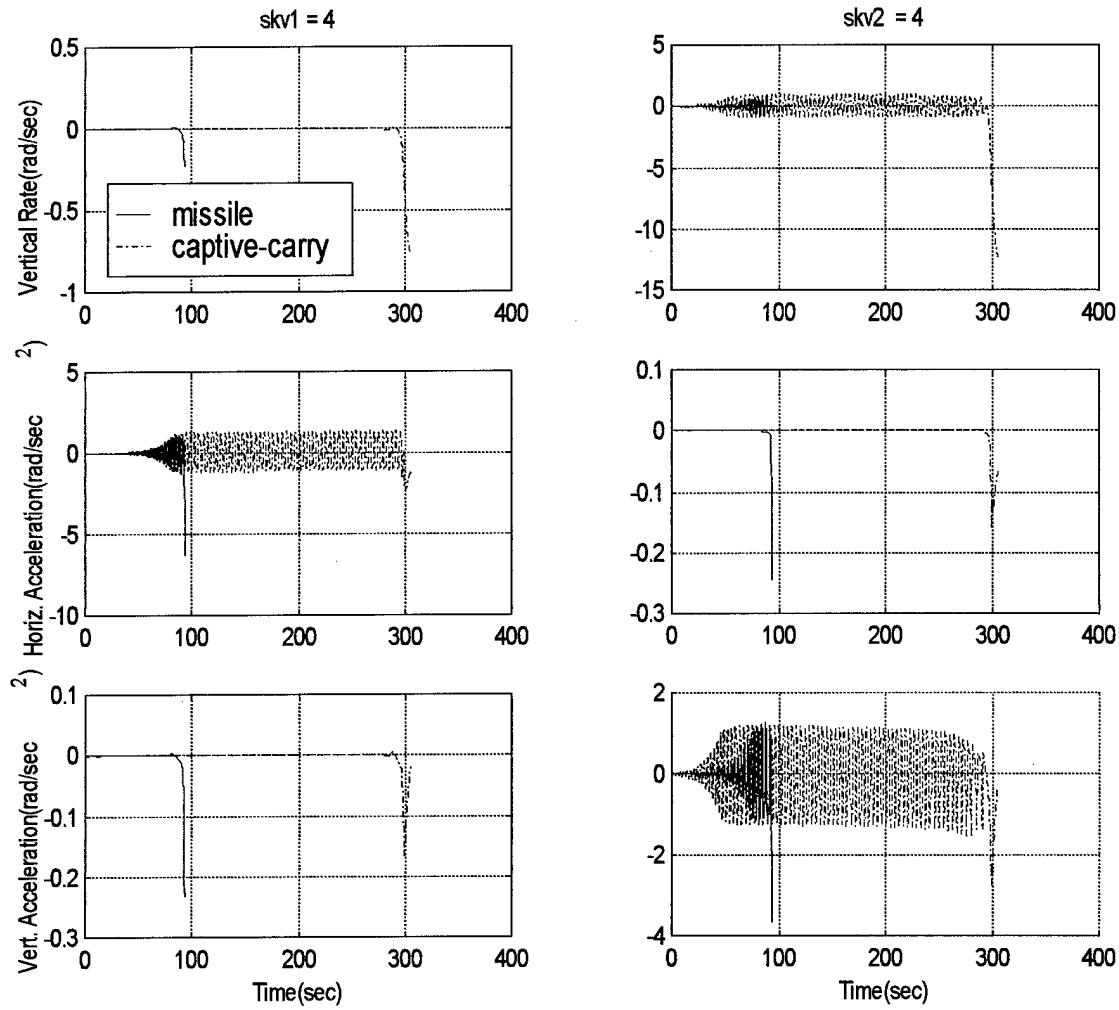


Figure 7.4. Seeker Response Comparison for $skv1$ and $skv2$

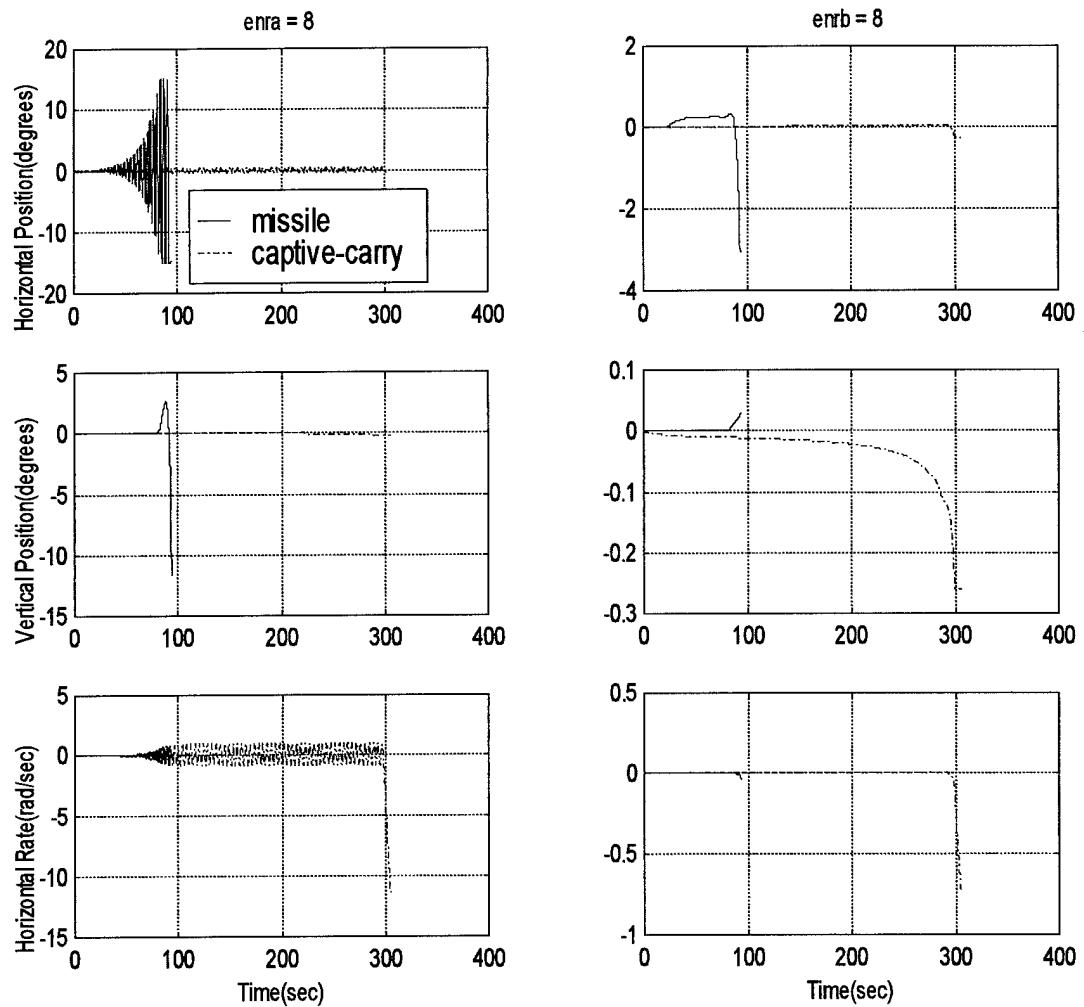


Figure 7.5. Seeker Response Comparison for $enra$ and $enrb$

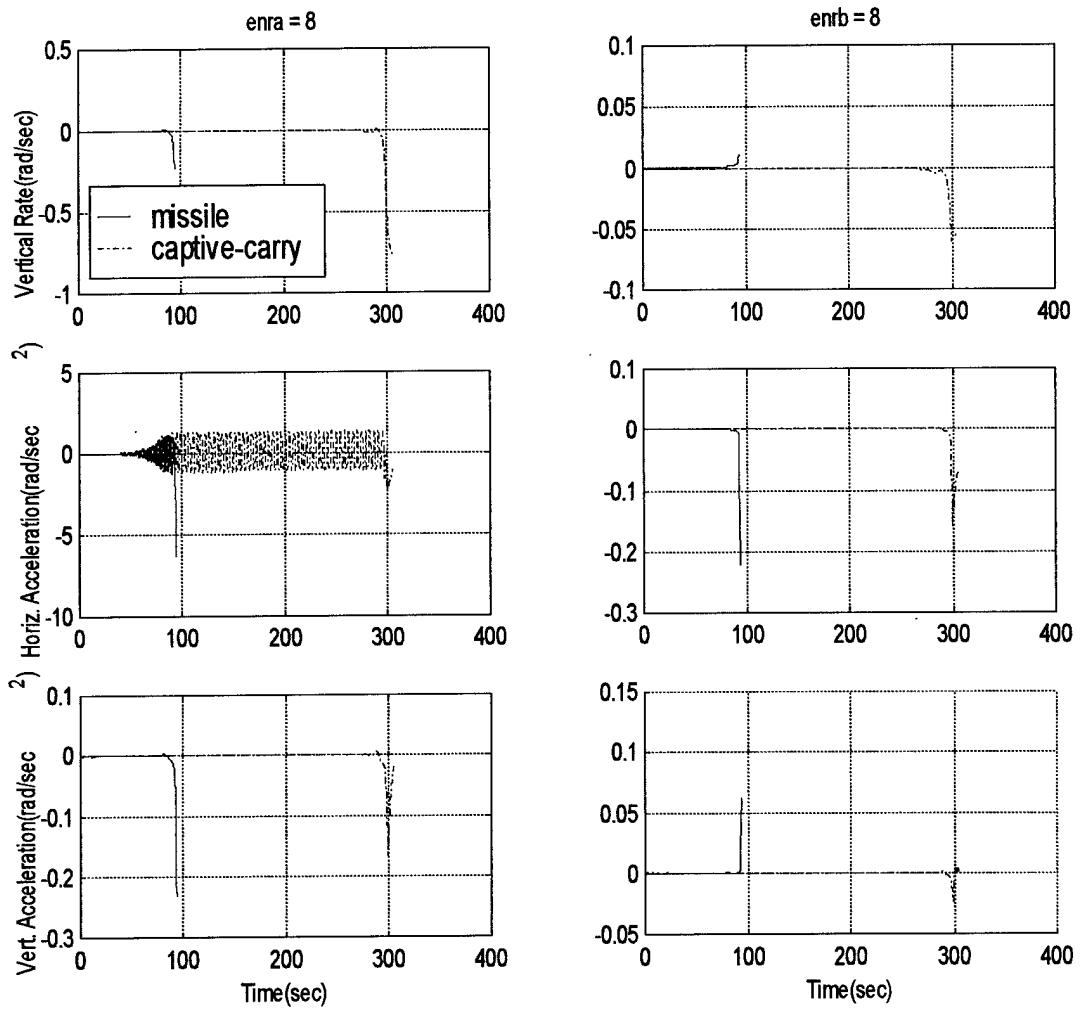


Figure 7.6. Seeker Response Comparison for $enra$ and $enrb$

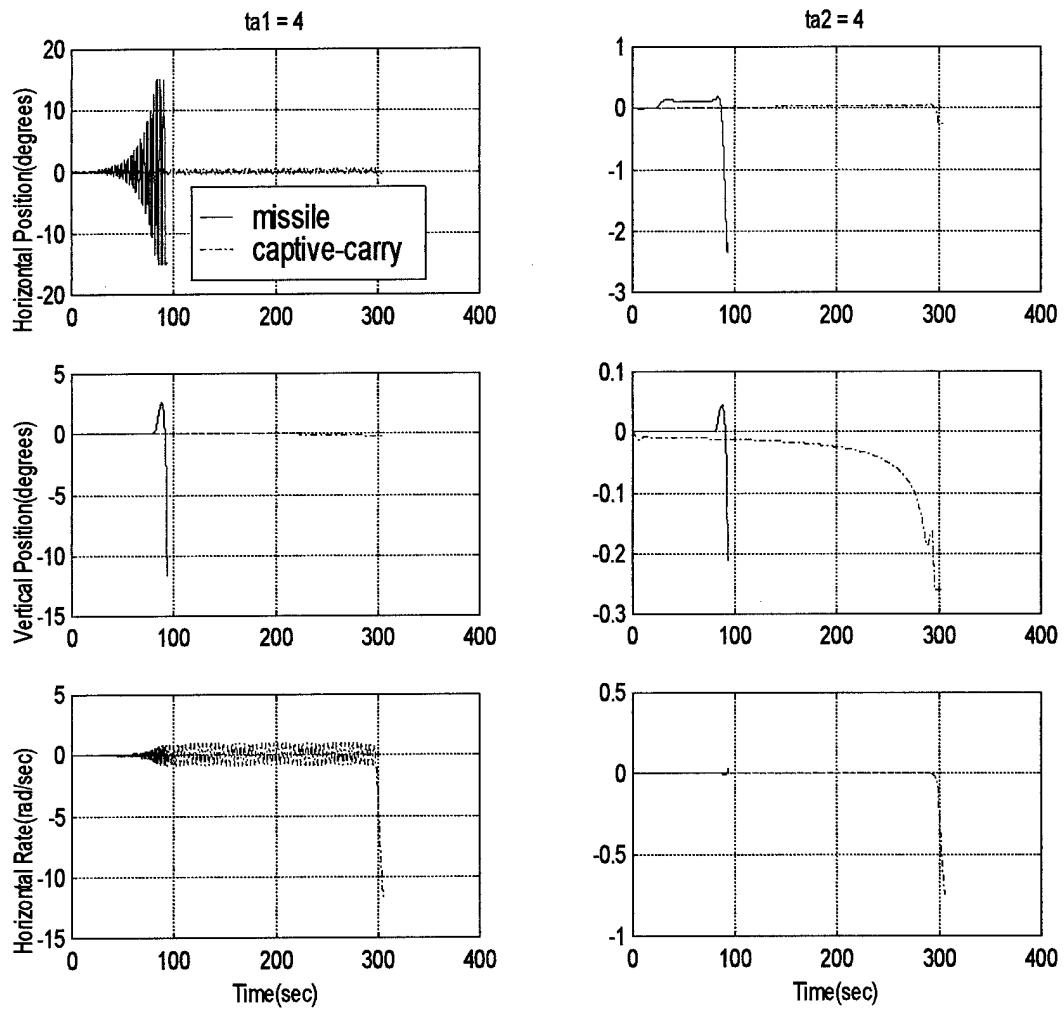


Figure 7.7. Seeker Response Comparison for $ta1$ and $ta2$

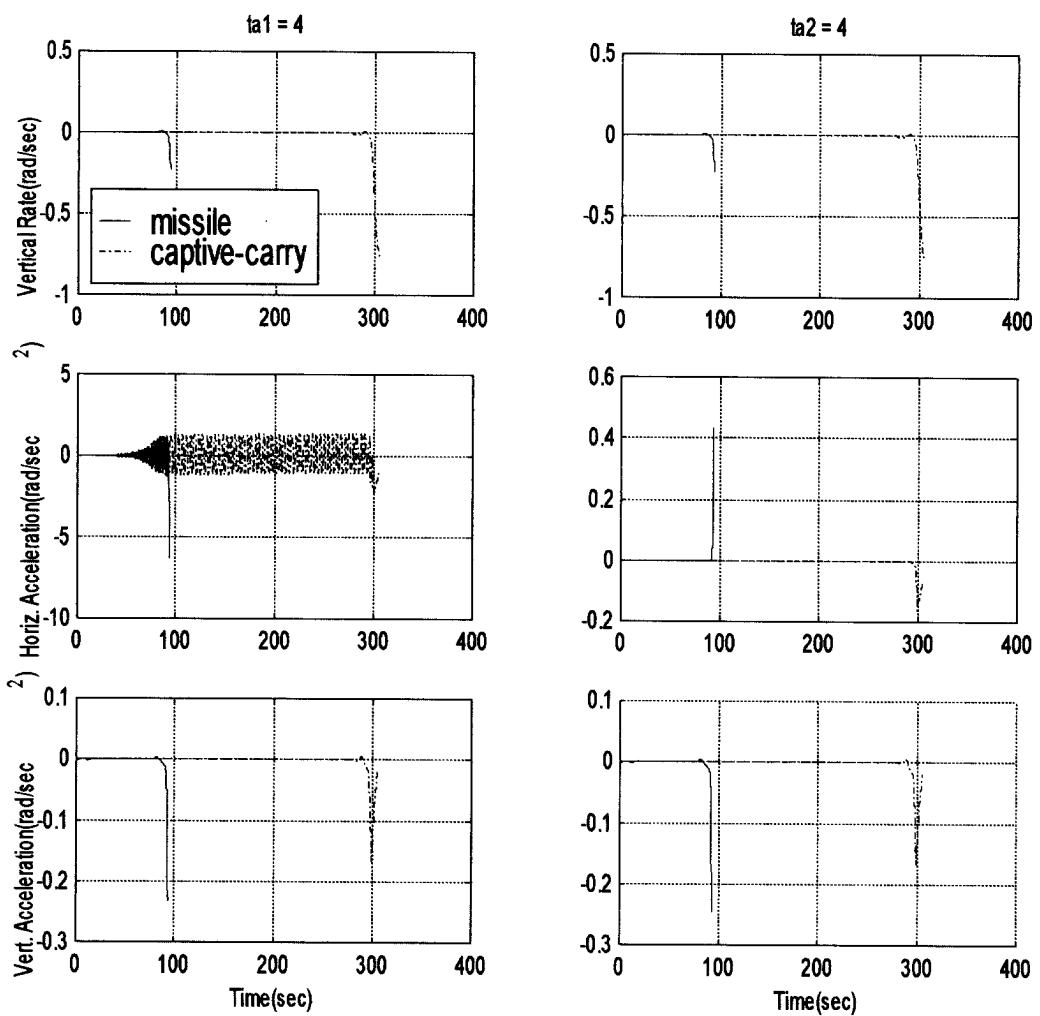


Figure 7.8. Seeker Response Comparison for $ta1$ and $ta2$

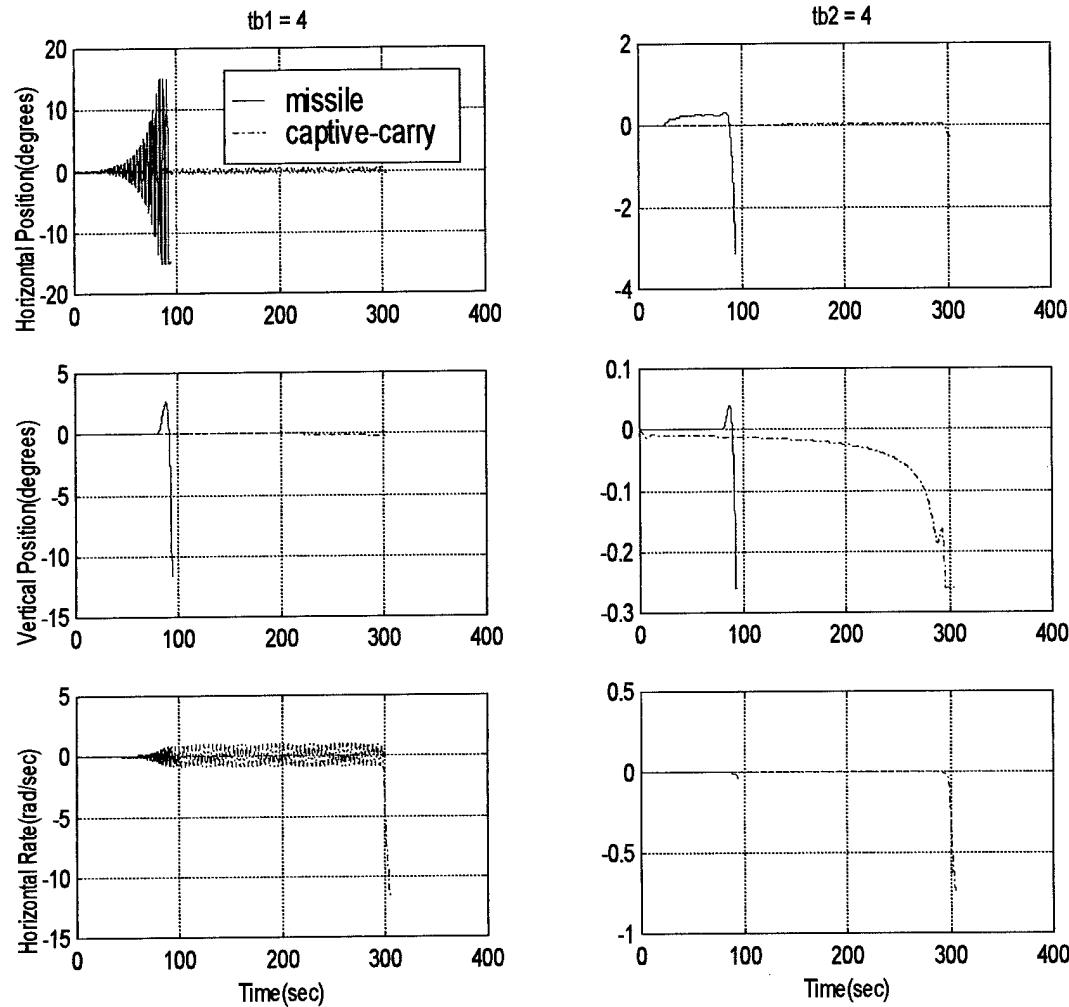


Figure 7.9. Seeker Response Comparison for $tb1$ and $tb2$

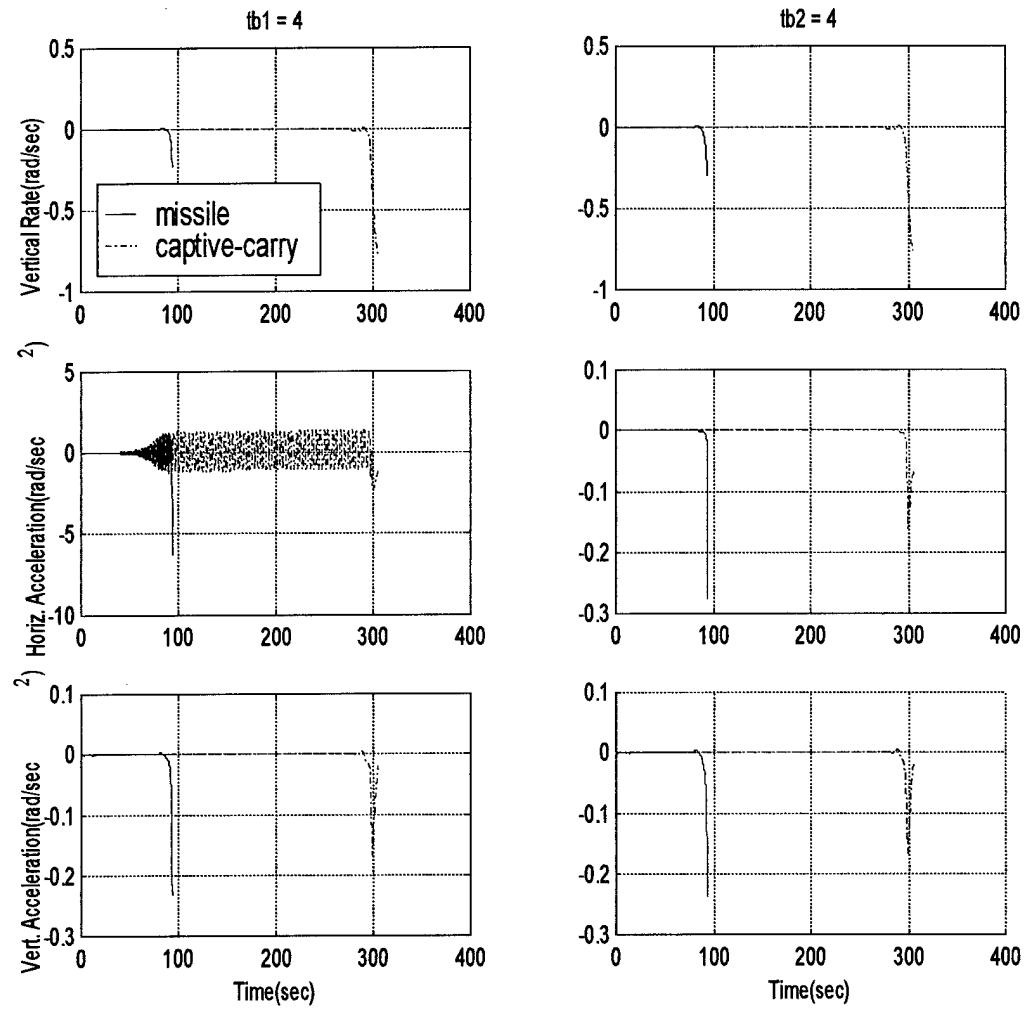


Figure 7.10. Seeker Response Comparison for $tb1$ and $tb2$

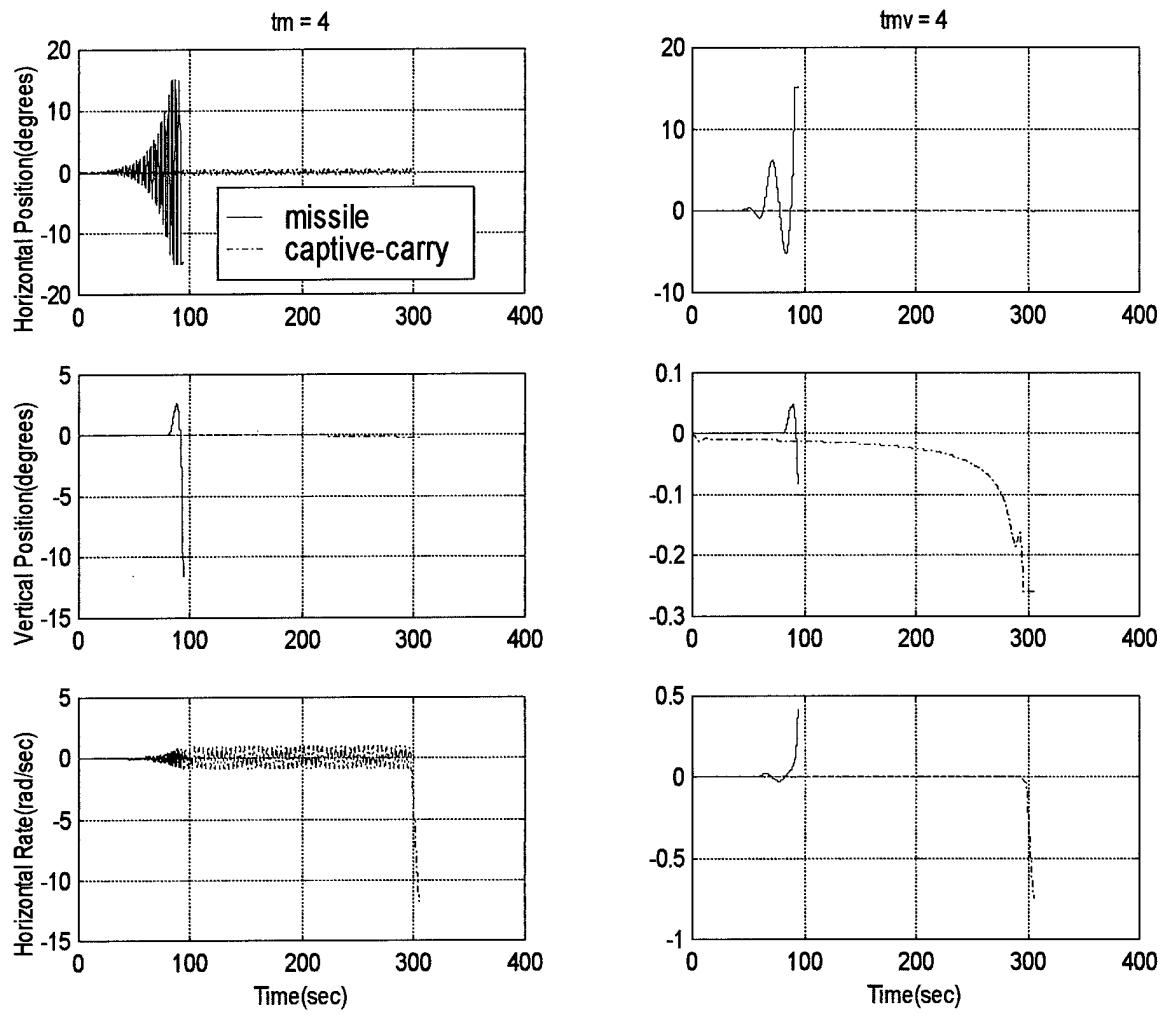


Figure 7.11. Seeker Response Comparison for tm and tmv

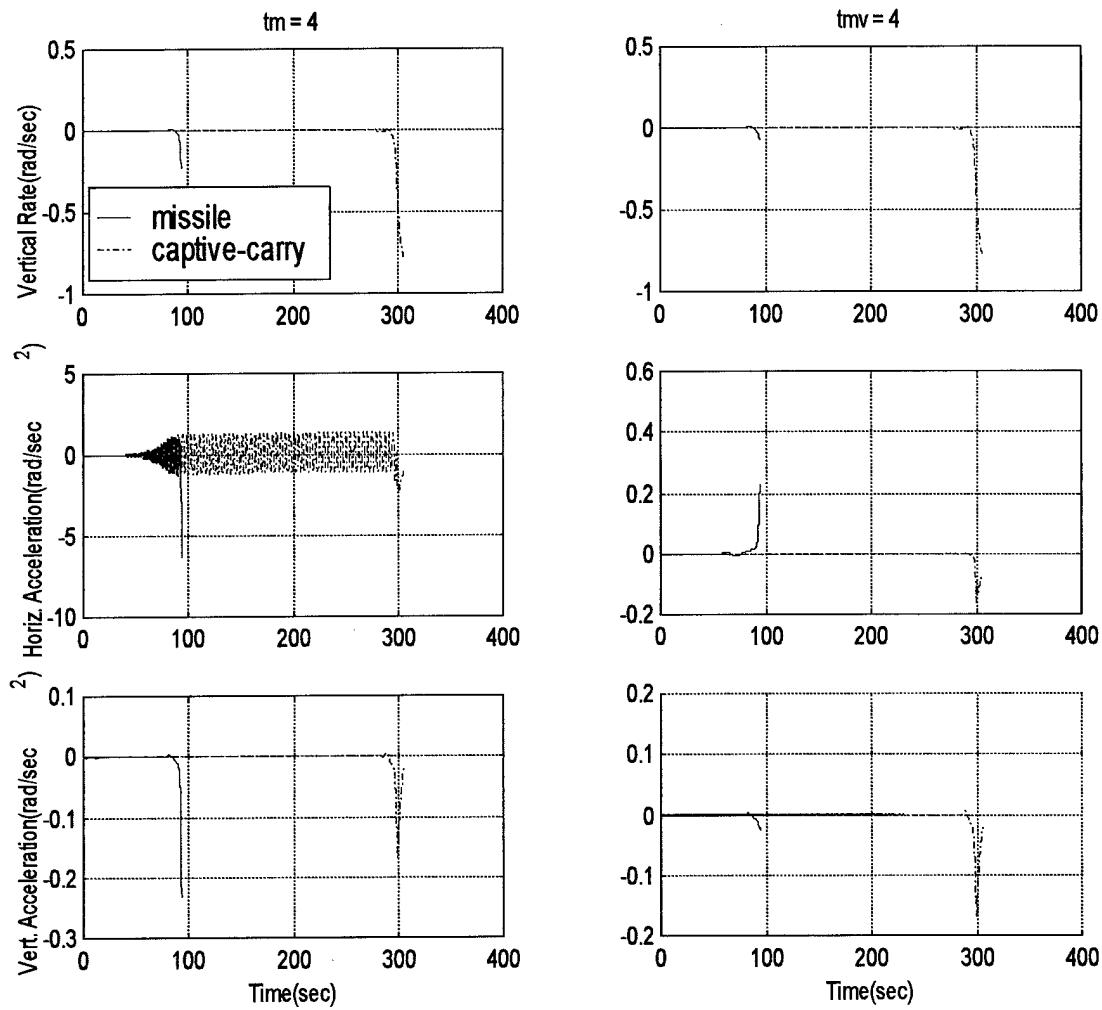


Figure 7.12. Seeker Response Comparison for tm and tmv

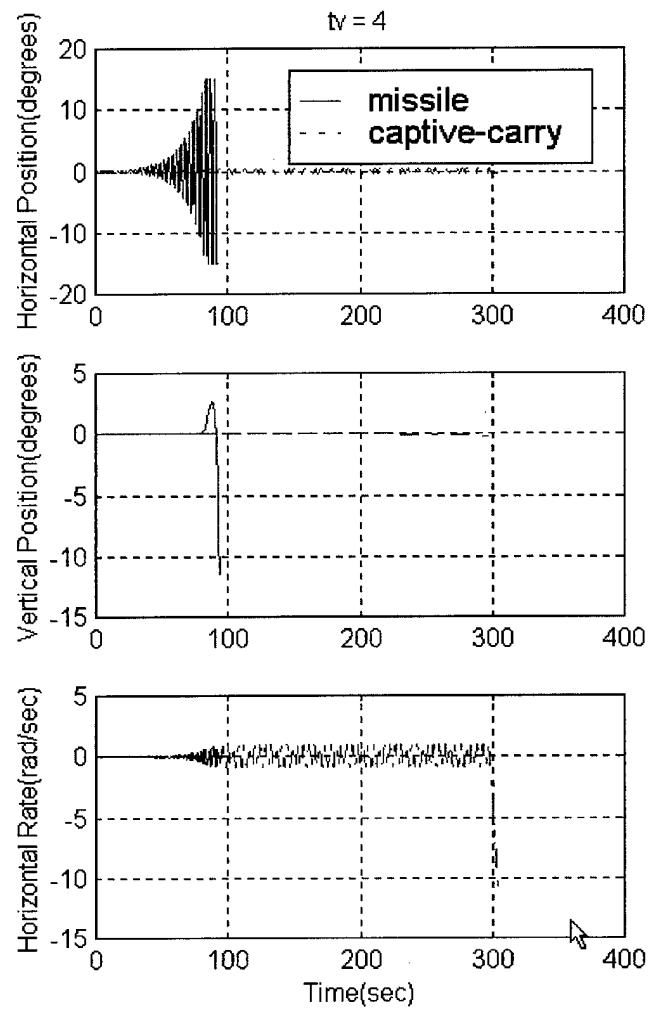


Figure 7.13. Seeker Response Comparison for tv

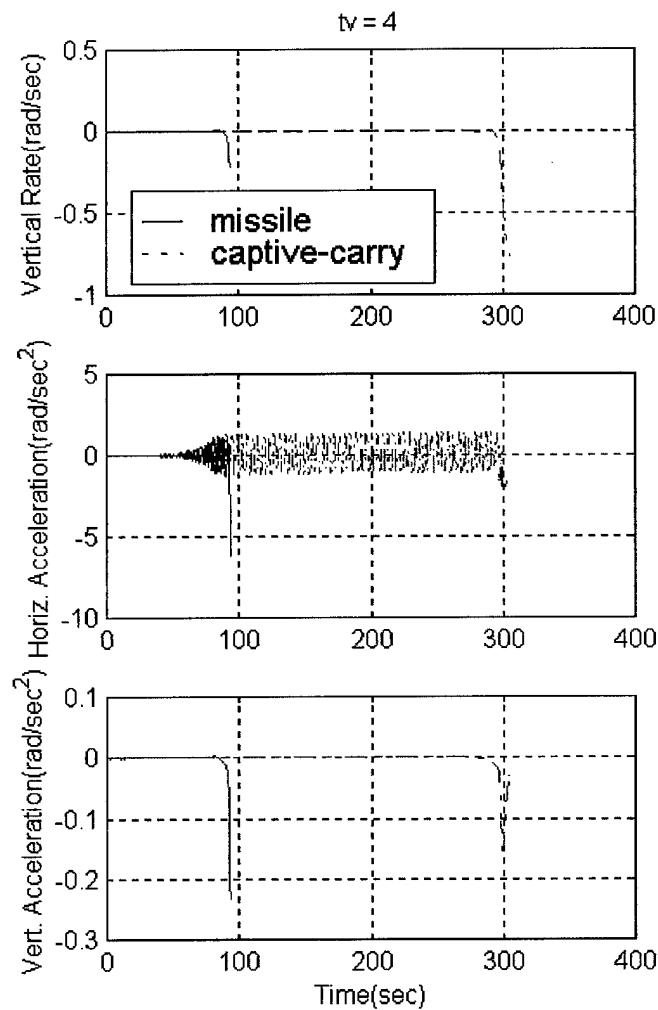


Figure 7.14. Seeker Response Comparison for $t\nu$

VIII. MISS DISTANCE PREDICTION FROM THE CAPTIVE-CARRY SIMULATION

A. INTRODUCTION

Chapter VII demonstrated that the seeker responses of the closed-loop and open-loop experiments have, in most of the simulations, similar results. This work combines the results of the missile and the captive-carry experiments to generate a single miss distance. In [Ref. 8] Neural Networks are used to predict the missile trajectory using only the seeker response obtained from the captive-carry results: range (R), azimuth (ϕ) and elevation (θ). Before being combined with the closed-loop results, the captive-carry results were pre-processed in order to adjust the data so that the differences in configurations were taken into account. In this work the captive-carry simulation results are processed by a trained neural network (NN) directly without any type of pre-processing to account for these configuration differences.

Neural networks, as defined in [Ref. 9],

(...) are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements. Commonly neural networks are adjusted, or trained, so that particular input leads to a specific target output (...).

Figure 8.1 shows a basic training process in a neural network.

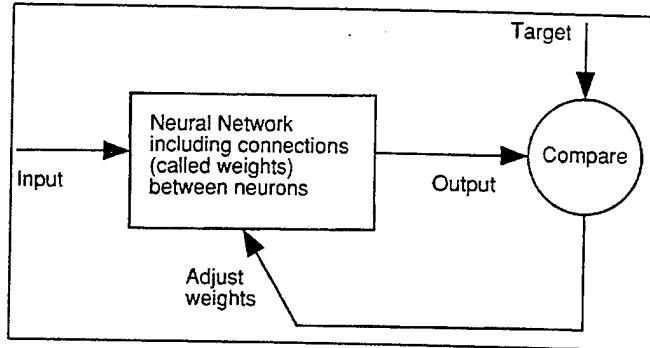


Figure 8.1. Neural Network Basic Training Process (After [Ref. 10])

In [Refs. 10 and 11] a good introduction to neural network theory is provided. In this paper, several steps were followed to reach our goal of generating the missile trajectory from the captive-carry seeker outputs. First, a simulation set was established, shown in Appendix N, with both the missile and captive-carry simulations. Those simulations were run using the ASCM Digital Model and *GUI Menu*, and the results from the two configurations were obtained. Second, after running the simulations, a neural network (NN) was created, its settings defined, and the NN trained. In the NN training, the weights and biases of the network are updated to get the desired target outputs (the actual missile position) from the missile seeker R , ϕ and θ inputs. The third step was testing the network with another actual missile simulation data set and comparing the NN output with the actual missile trajectory. Finally, when the neural network was trained and tested, the captive-carry seeker outputs (R , ϕ and θ) were introduced as inputs to the NN to generate a missile trajectory prediction.

B. ASCM DIGITAL MODEL CONFIGURATION FOR THE NEURAL NETWORK

Appendix N shows the spreadsheet that contains the simulations used to establish the tactical scenario for the open and closed-loop simulations. The tactical scenario is as follows:

- The target is the Original Target (*swtsh* = 1), the Nulka decoy is activated (*nlk* = 1), the threat is a missile (*carry* = 1), the missile seeker is monopulse (*seltrac* = 1) and there is noise in the selected seeker (*noise* = 1)
- The other ASCM Digital model parameters are set to the default values
- The noise in the monopulse seeker is the random factor among the 100 simulations run

The *initial.m* code was slightly modified in order to store the new values of the model parameters. Also, the GUI *Menu* was modified to store the simulation results in a distinct matrix (*goldxx.mat*).

C. TRAINING AND TESTING THE NEURAL NETWORK

This section discusses the required steps to establish a neural network (NN) that is able to predict the missile dynamics from the captive carry experiment. In the first step, the neural network is designed and shown Figure 8.2.

Where:

N = number of inputs or features (N=3, R, ϕ and θ)

p = input matrix

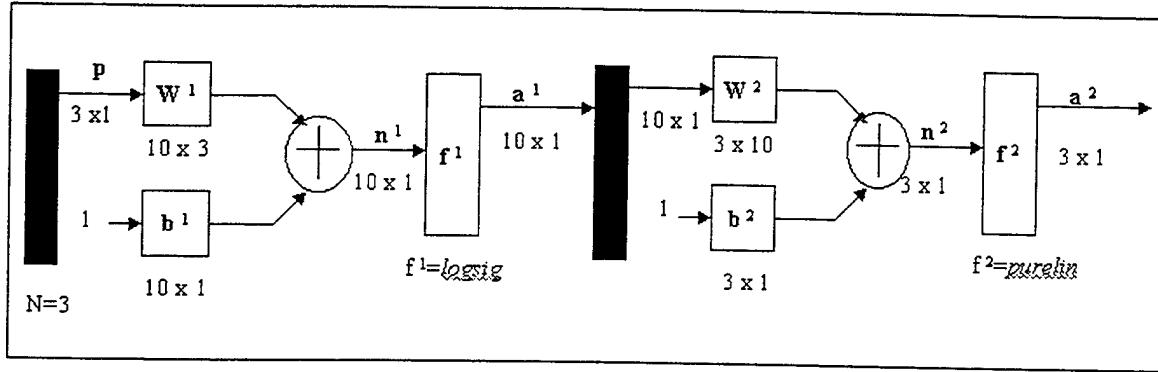


Figure 8.2. Neural Network Configuration for Missile Dynamics Prediction

b = neuron biases vector

W = weight matrix

n = result from the operation $\mathbf{W}\mathbf{p} + \mathbf{b}$

f = transfer function (in our NN, the first layer has a log sigmoid (*logsig*) and the second layer has a linear (*purelin*) transfer functions)

a = neural network layer output

The number of neurons in the first layer is 10 and 3 in the second layer. The NN was set to feedforward. There is no feedback of the NN outputs to the NN inputs. The *superscripts* 1 and 2 in Figure 8.2 indicate that the neural network chosen for our prediction has two layers. It was decided that the acceptable error for weight convergence was 10^{-6} .

For the NN training, a supervised learning rule is used. The learning rule (the procedure for modifying the weights and biases of the network) uses R , ϕ and θ for each time increment in the 45 odd missile simulations shown in Appendices N and O. The outputs were the corresponding missile positions. Due to the non-linear nature of the inputs and outputs, the first layer of the NN has a non-linear transfer function called log

sigmoid (*logsig* function in MATLAB[©]) as shown in Figure 8.3. A linear transfer function (*purelin* function in MATLAB[©]) is used for the second layer as shown in Figure 8.4. The Levenberg-Marquardt (LM) training algorithm is used (*trainlm* in MATLAB[©]) since it is considerably faster than backpropagation. This algorithm reduces the mean square error set of 10^{-6} or a selected number of iterations (500). The training and testing MATLAB[©] codes for the NN are shown in Appendix O. The minimum error is reached after 14 epochs for the first training set, 14 also for the second, and zero for the third.

The NN testing is done using the odd missile simulations 5, 15, 25, 35, 45, 55, 65 and 75 shown in Appendix N. The NN results are presented in pairs by Figures 8.5-6, 8.7-8, 8.9-10, 8.11-12, 8.13-14, 8.15-16, 8.17-18, 8.19-20. The first figure of each pair presents the seeker inputs to the NN from the close-loop experiment and the second graph in the pair presents the comparison between the real and predicted missile trajectory and the mean square error between the two trajectories. Table 8.1 presents the miss distance obtained from the missile simulation and the NN prediction.

D. MISSILE TRAJECTORY PREDICTION USING THE NEURAL NETWORK

After training and testing the NN using the R , ϕ , θ , values from the missiles seekers as input, and the missile position as the desired output, the NN is now ready to be used to predict the missile trajectory using the R , ϕ , θ , values from the captive-carry seeker. Those values can be used as inputs to the NN and are expected to give valid

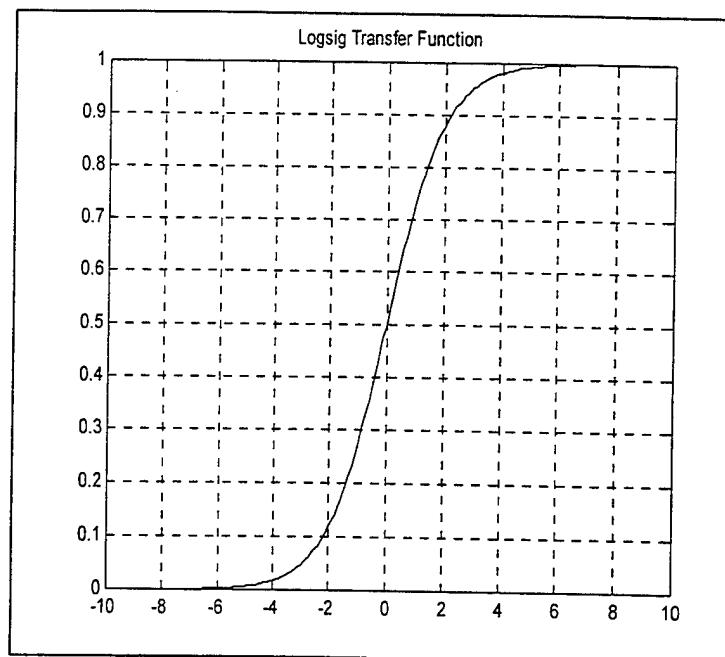


Figure 8.3. Log Sigmoid Transfer Function

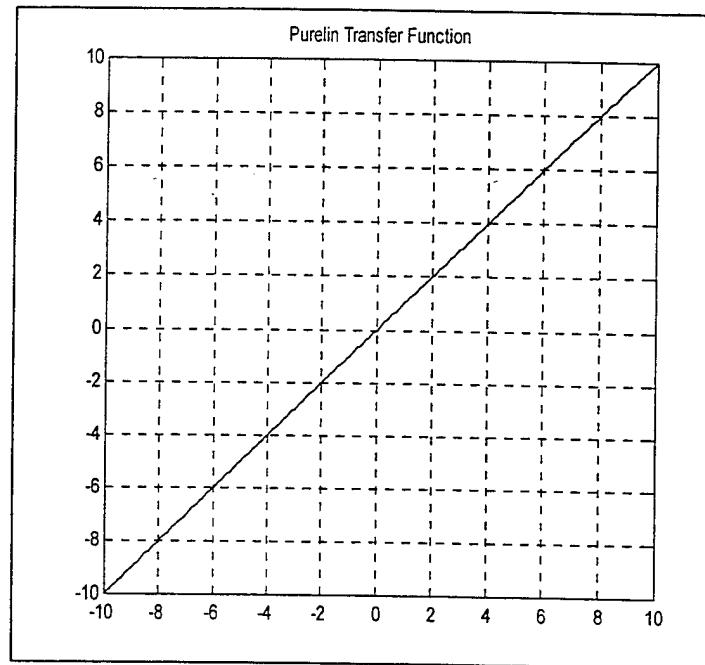


Figure 8.4. Linear Transfer Function

outputs since their range is similar to the range of the training and testing date set except for the fact that the configurations are different. Simulations 8, 18, 28, 38, 48, 58, 68 and 78 from Appendix N and the code used to simulate the missile trajectory are shown in Appendix O.

The results are presented in two graphs. The first graph (Figure 8.21) is the captive-carry seeker inputs to the NN. The second graph presents the captive-carry trajectory, the neural network prediction, the actual missile trajectory associated with that simulation and the Original Target trajectory. As shown in Figure 8.22, the NN predicts the miss trajectory quite well in the beginning. However, by the down range of 60,000 ft, the prediction tries to track the captive-carry trajectory. In the X-Y plane the NN corrects the prediction back to the missile trajectory. In the Z-direction, the NN outputs are influenced by the differences in altitude between the open-loop (1000 ft.) and closed-loop experiments (sea-skimming). The NN prediction starts to follow the captive-carry altitude values. Since the closed and open-loop simulation frame times are different in function from the threat velocities, only 97.22 % of the captive-carry data set was used in order to put the results on a same scale to allow comparisons between them. Only about 84 data points (2.78 %) of the captive-carry experiment was not used by the NN to calculate the missile trajectory prediction. Each simulation generates about 3052 points.

Finally, Table 8.2 presents the comparison between the missile and the predicted miss distance from the captive-carry experiment. The results are very impressive considering that they were obtained from the captive-carry experiment seeker responses and, only because the aircraft altitude is strongly out of the missile altitude, more accurate

results could not be obtained. For further improvements in the NN, the possibility of using feedback NN may result in a better miss distance prediction. For now, it is suggested that the aircraft altitude be simply subtracted from the result of the NN as it is shown in Table 8.2 to get a “corrected” value of the NN altitude. After the new miss distance calculation, it is possible to visualize that the predicted miss distance values range is very close to the ones obtained from the closed-loop experiments.

Simulation Number	Actual Miss Distance (ft)	Predicted Miss Distance (ft)
5	1,086.9	1,568.9
15	1,0868.8	1,566.5
25	1,088	1,5650
35	1,086.8	1,581.5
45	1,094.6	1,526.6
55	1,098.6	1,723
65	1,086.9	1,566.6
75	1,086.5	1,574.1

Table 8.1. Miss Distance Comparison Between The Actual Threat And The NN Prediction Using Closed-Loop Experiment Information

Simulation Number	Actual Miss Distance (ft)	Predicted Miss Distance (ft) ("Corrected")
8	1,086.9	2,019.2
18	1,087	2,019.2
28	1,086.4	2,019
38	1,086.8	2,019.3
48	1,052	2,023.8
58	1,051.2	2.020
68	1,086.5	2,019.2
78	1,086	2,019.2

Table 8.2. Miss Distance Comparison Between The Actual Threat And The NN Prediction Using Open-Loop Experiment Information

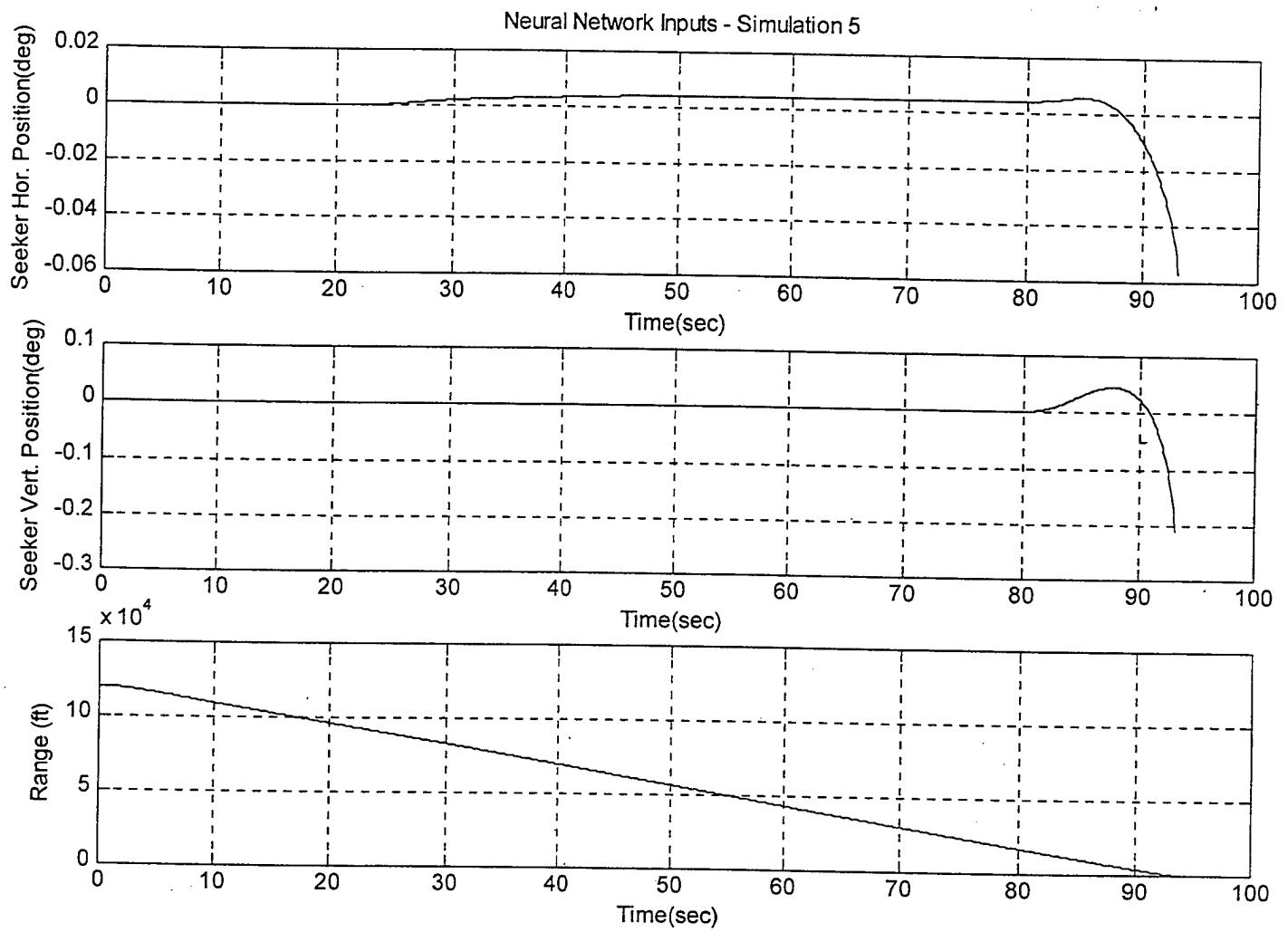


Figure 8.5. Seeker Inputs for the Neural Network (Simulation 5)

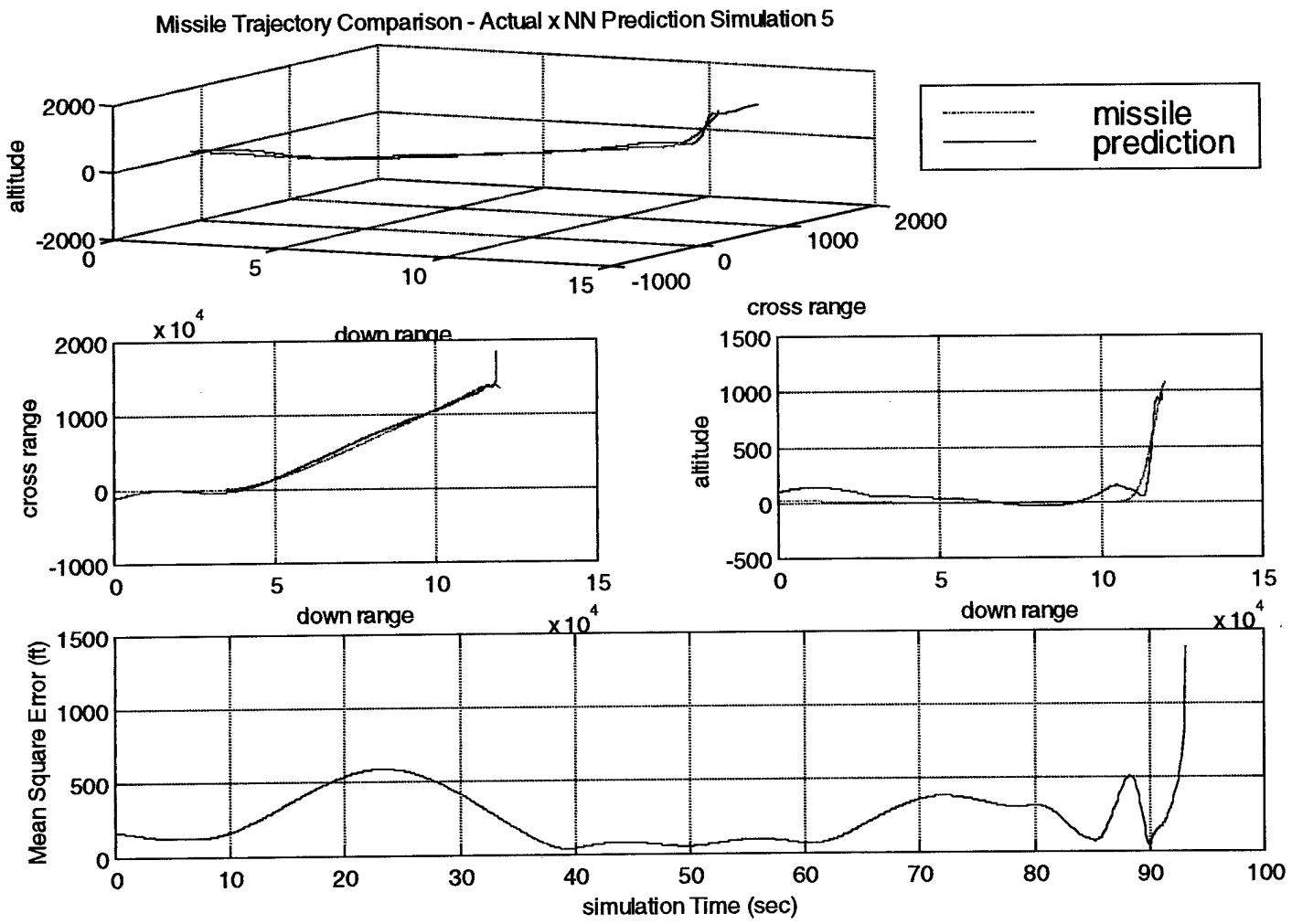


Figure 8.6. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 5

Neural Network Inputs - Simulation 15

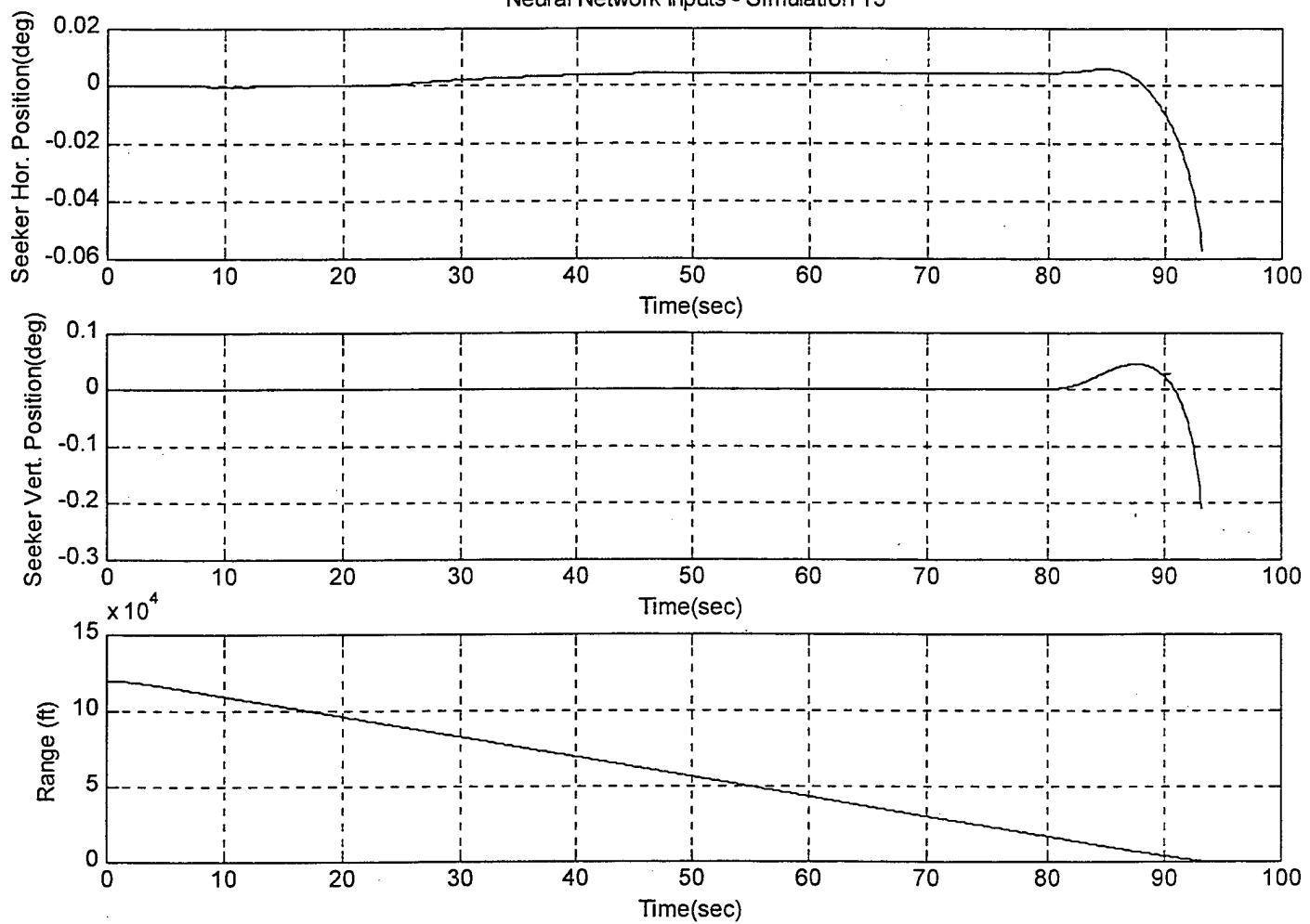


Figure 8.7. Seeker Inputs for the Neural Network (Simulation 15)

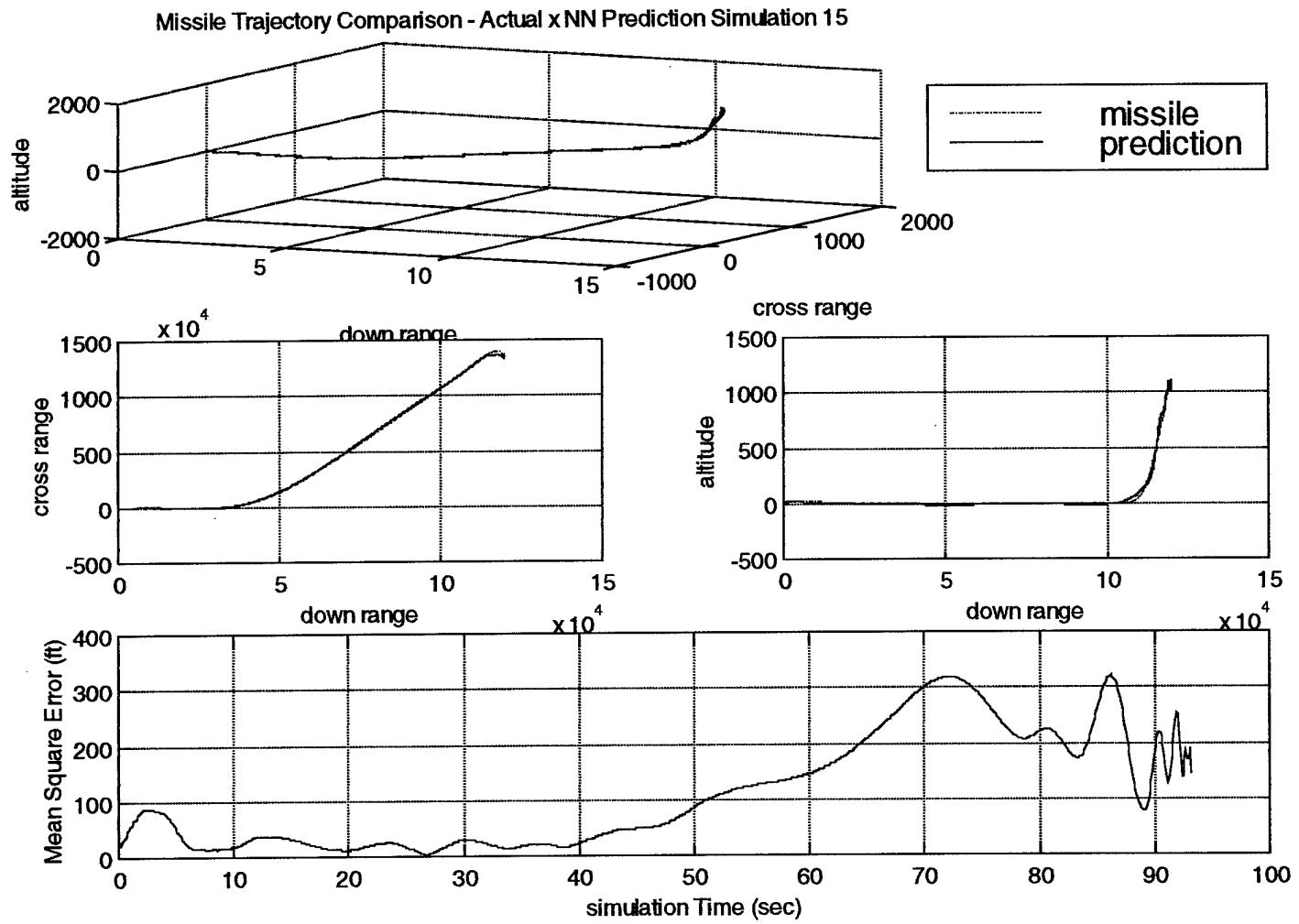


Figure 8.8. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 15

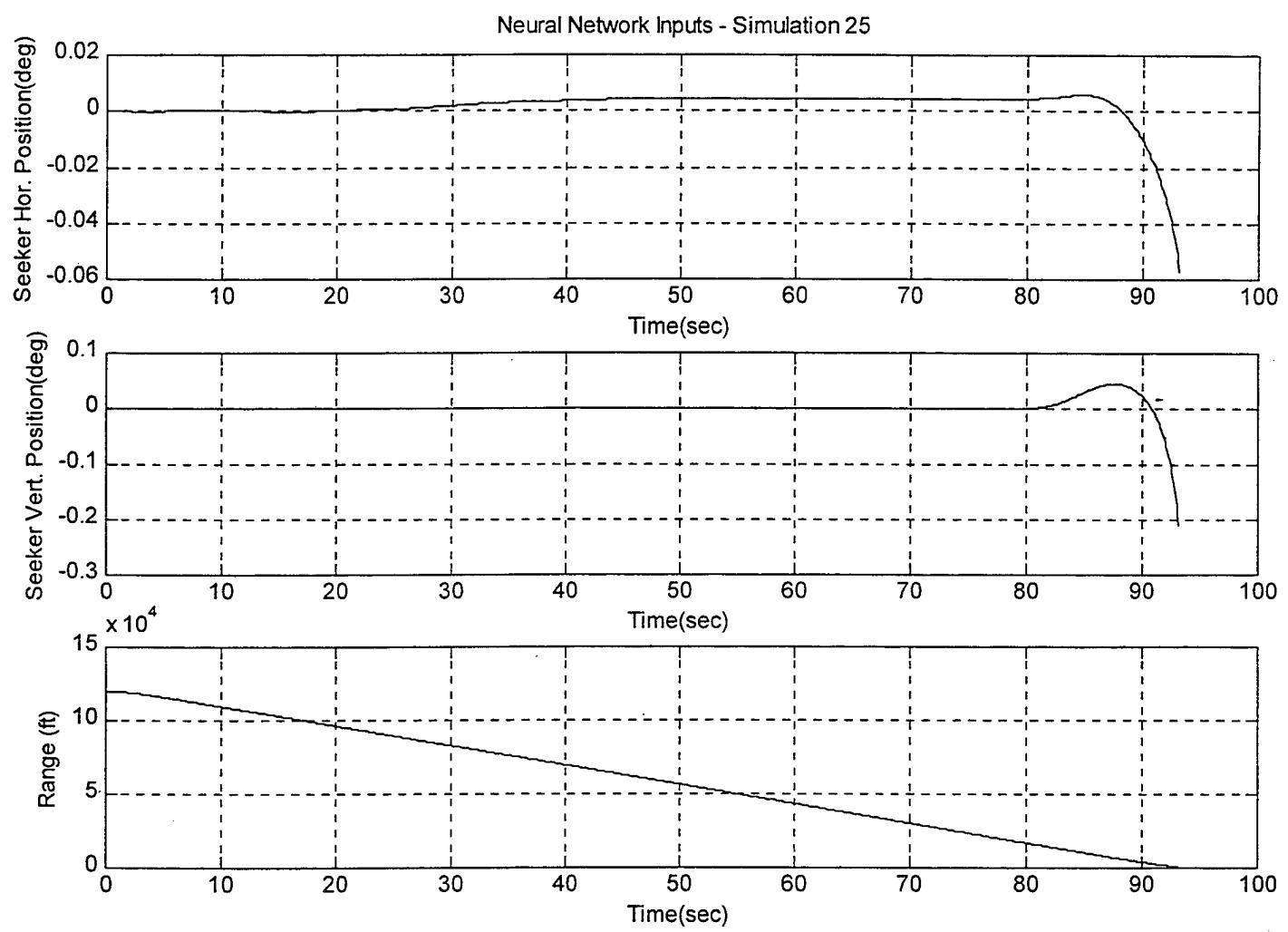
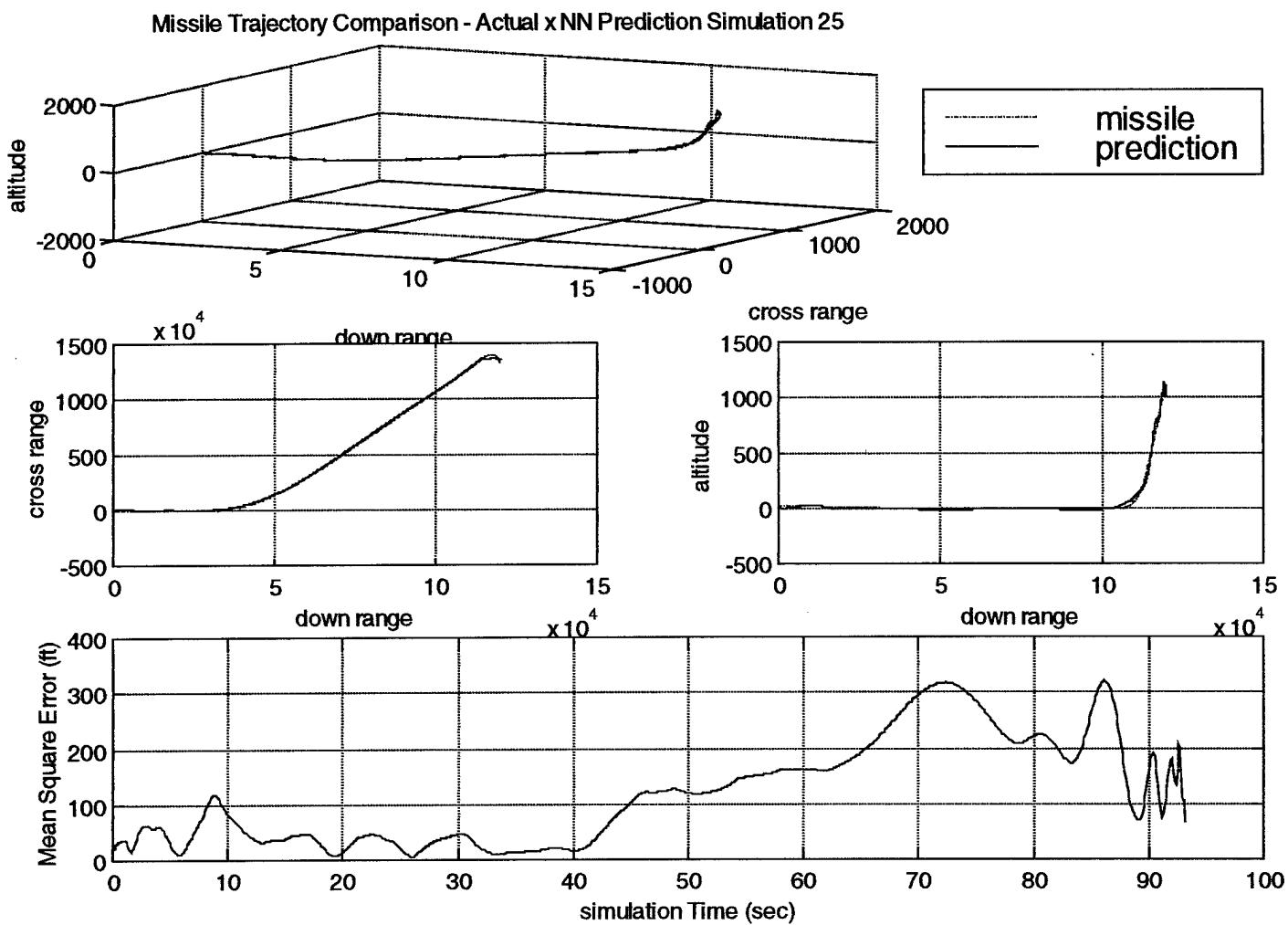


Figure 8.9. Seeker Inputs for the Neural Network (Simulation 25)



**Figure 8.10. Missile Trajectory Comparison – Actual x NN Prediction-Simulation
25**

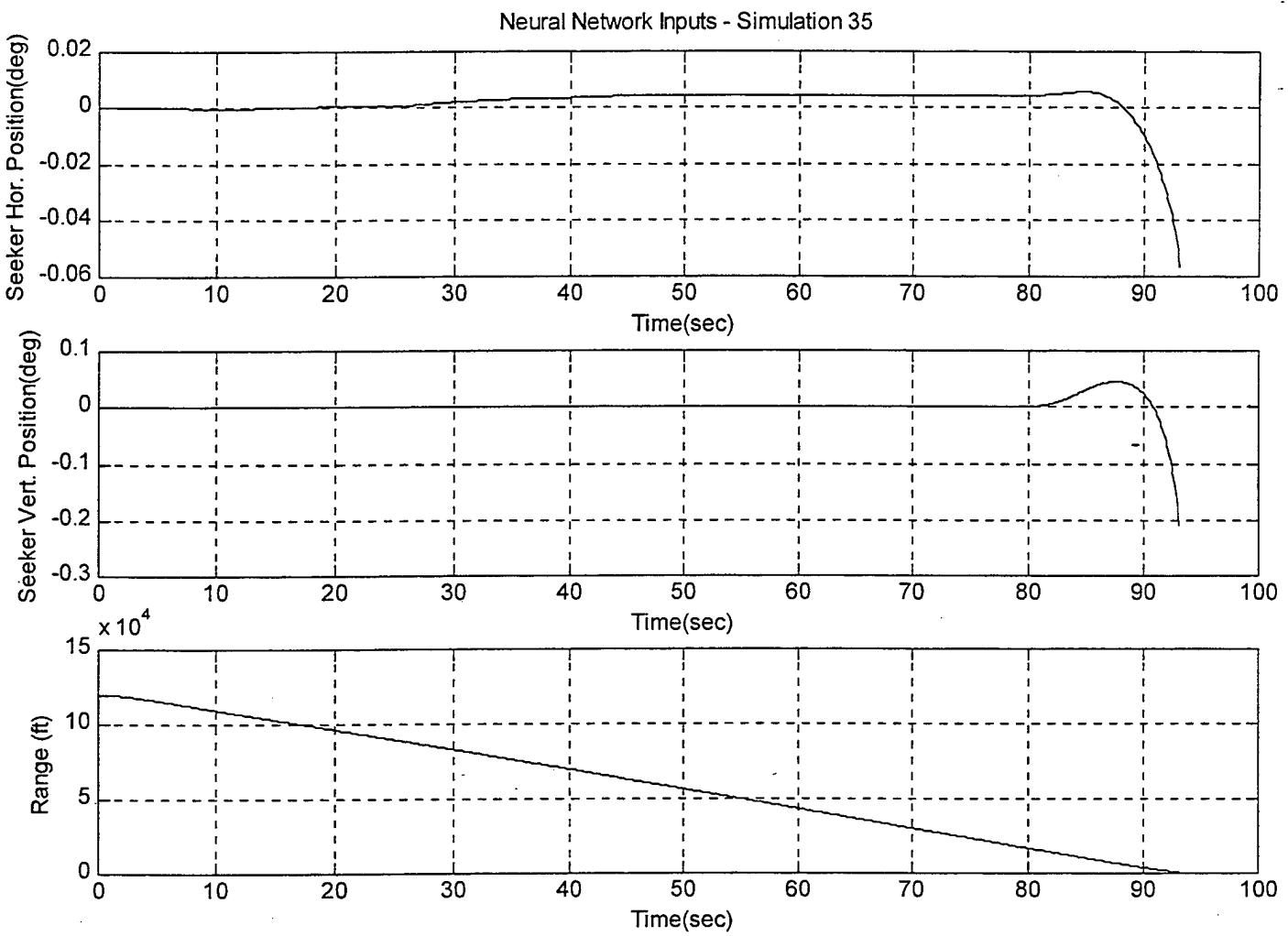


Figure 8.11. Seeker Inputs for the Neural Network (Simulation 35)

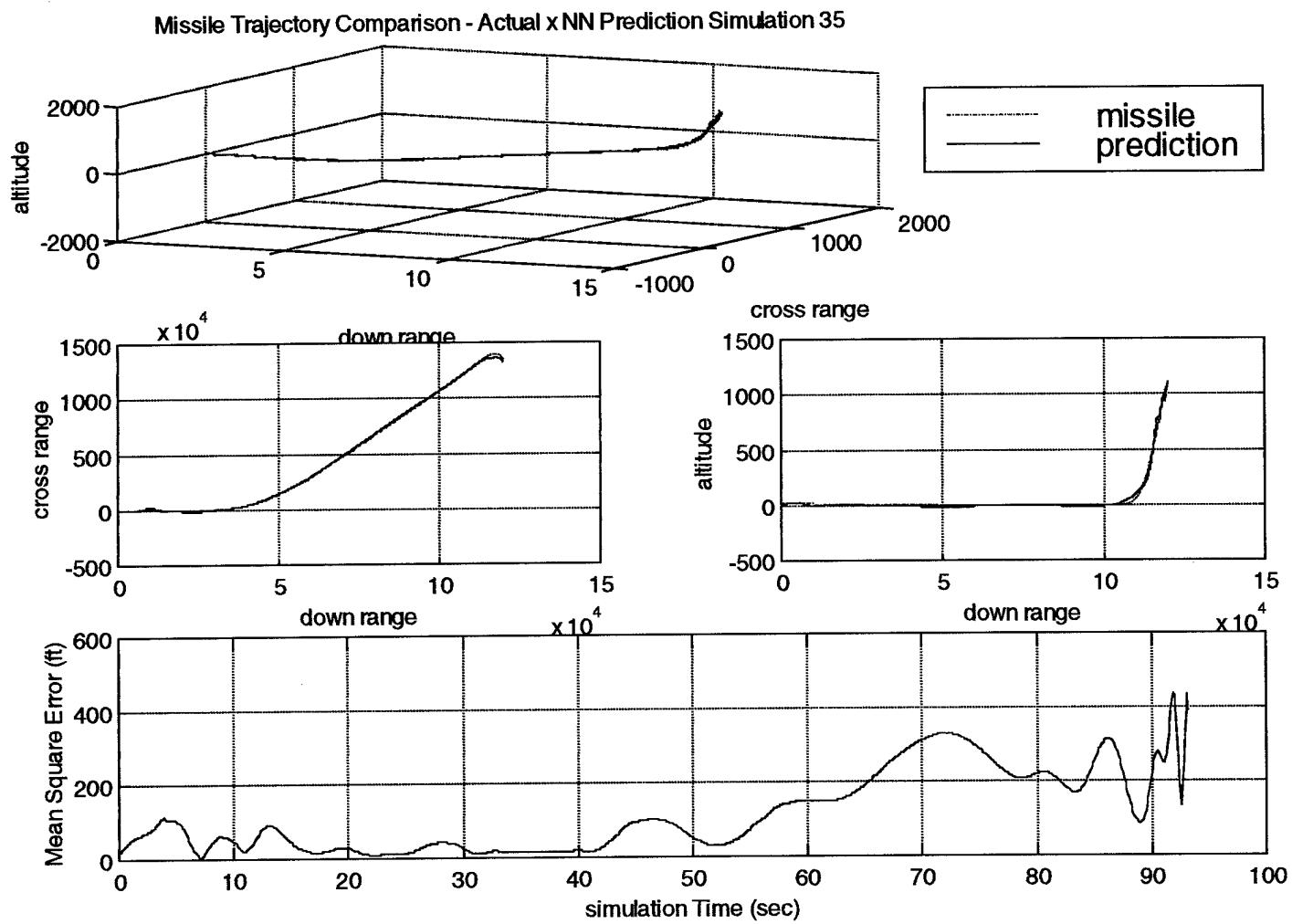


Figure 8.12. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 35

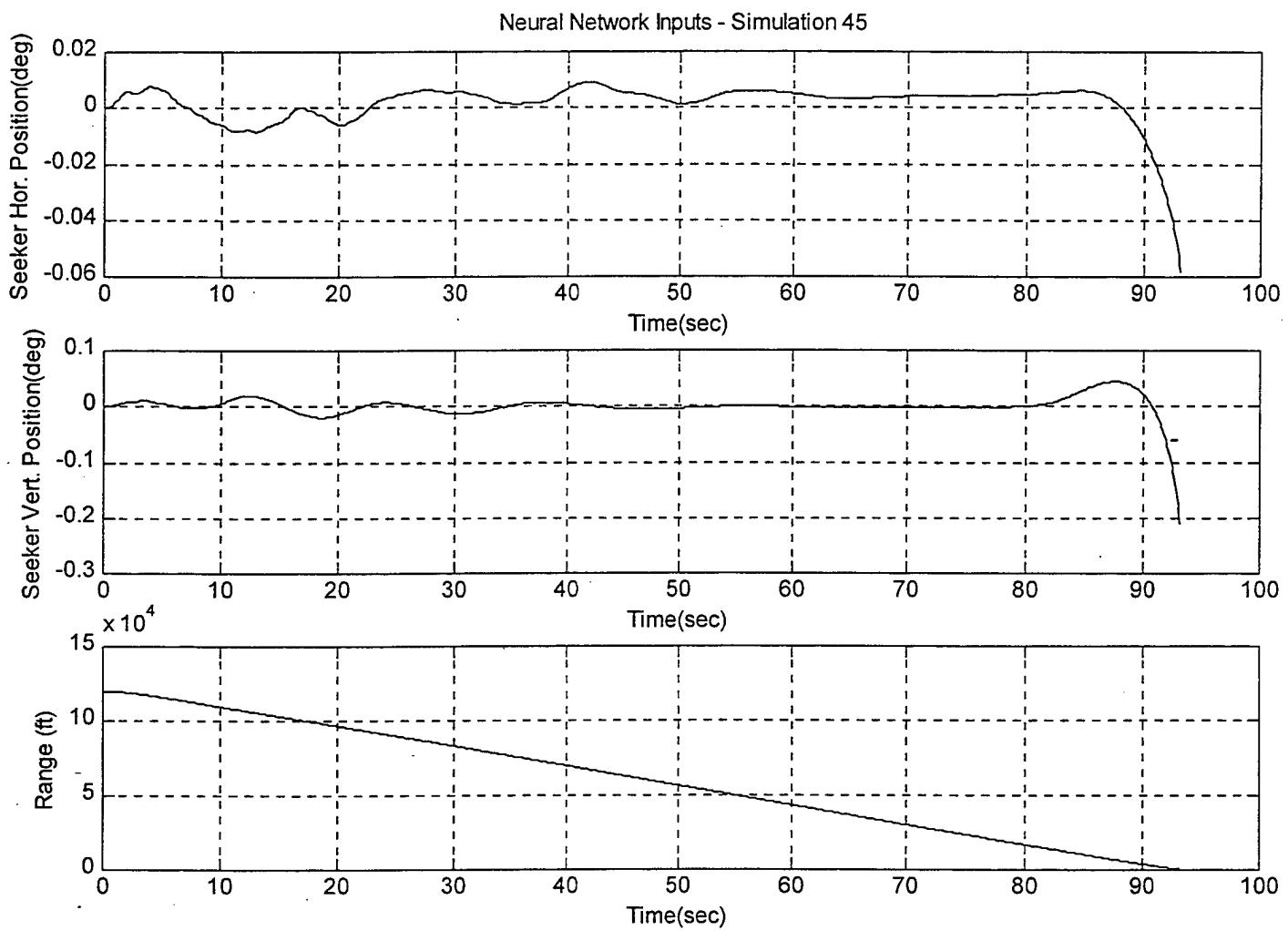


Figure 8.13. Seeker Inputs for the Neural Network (Simulation 45)

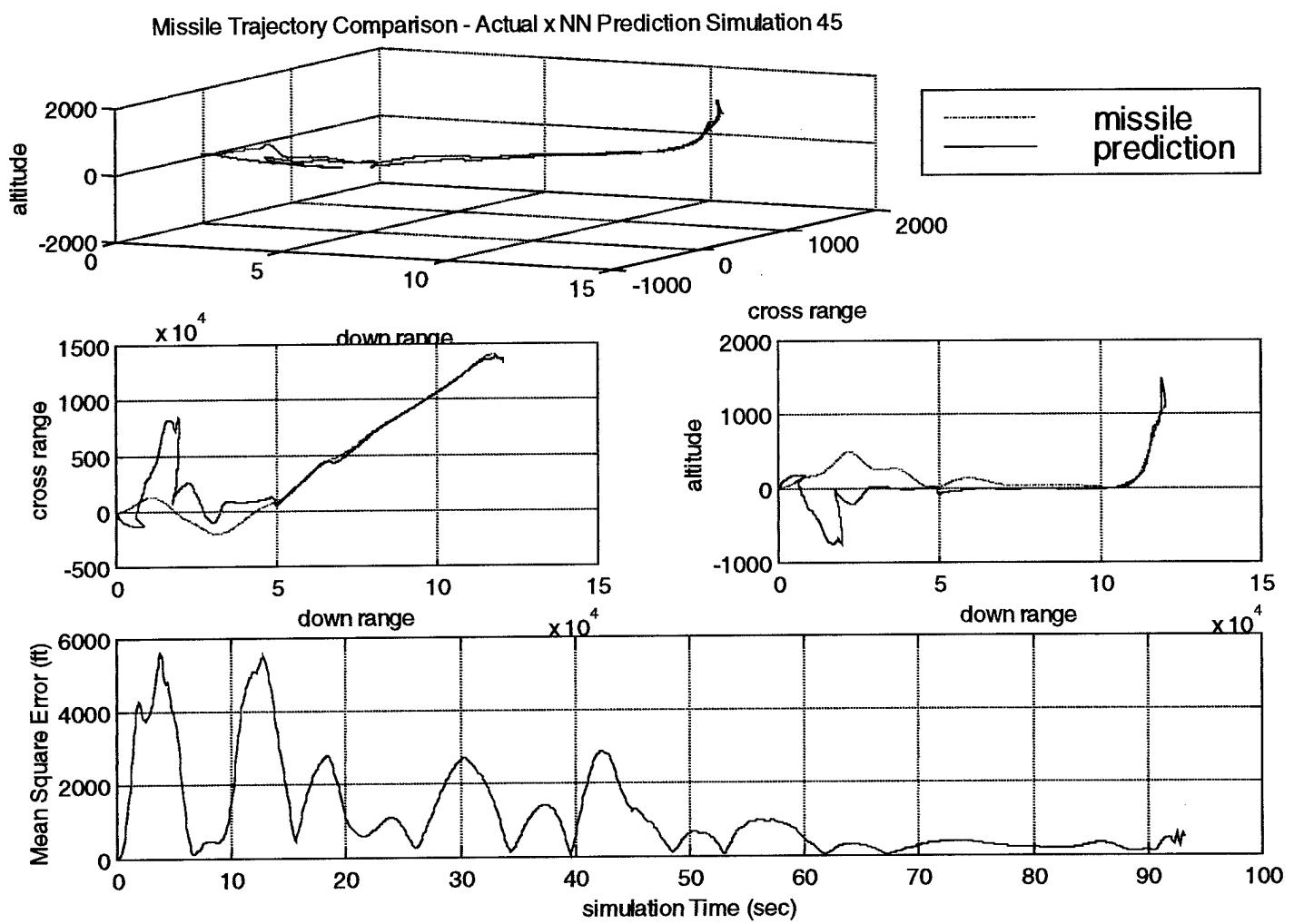


Figure 8.14. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 45

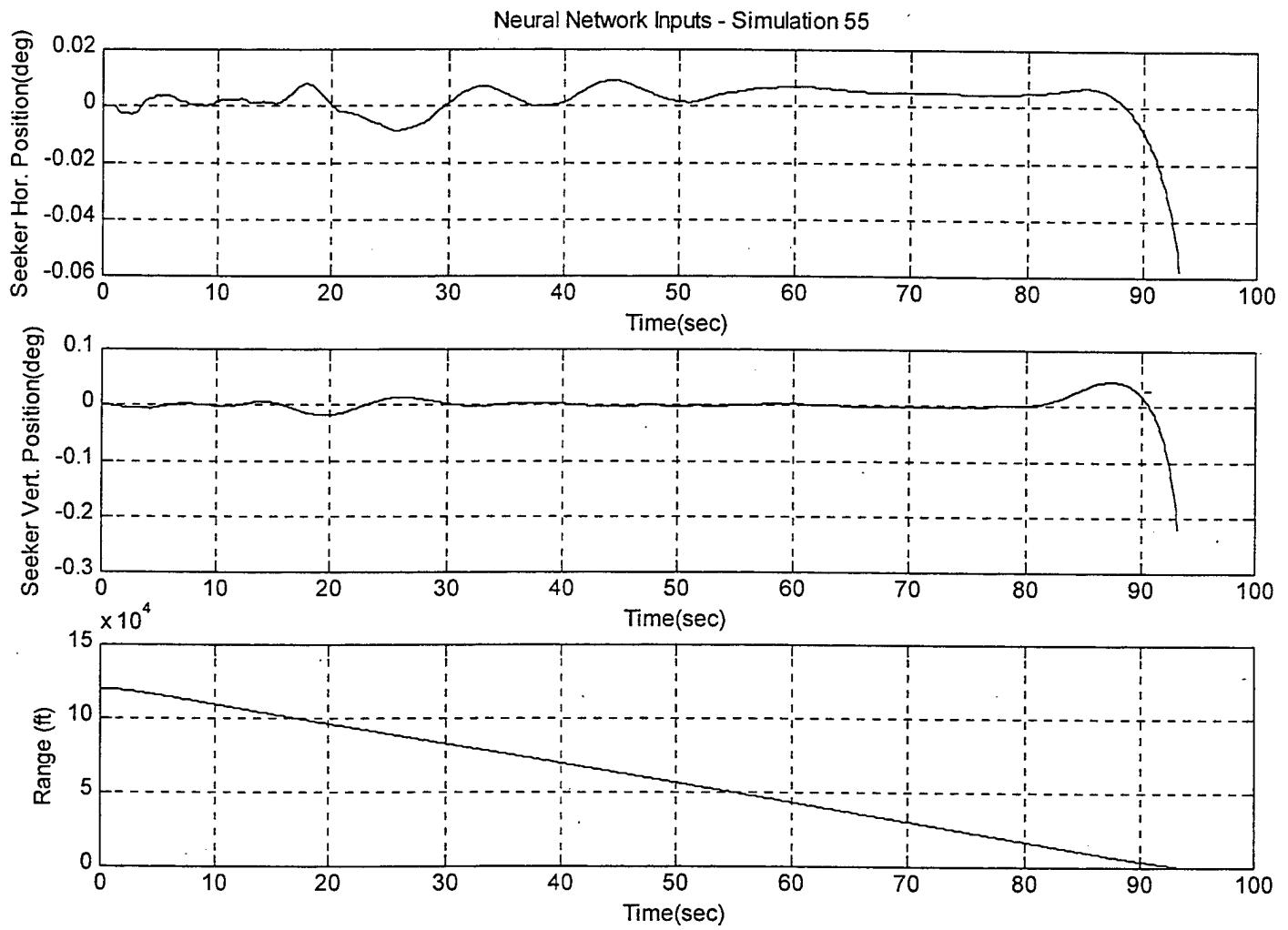


Figure 8.15. Seeker Inputs for the Neural Network (Simulation 55)

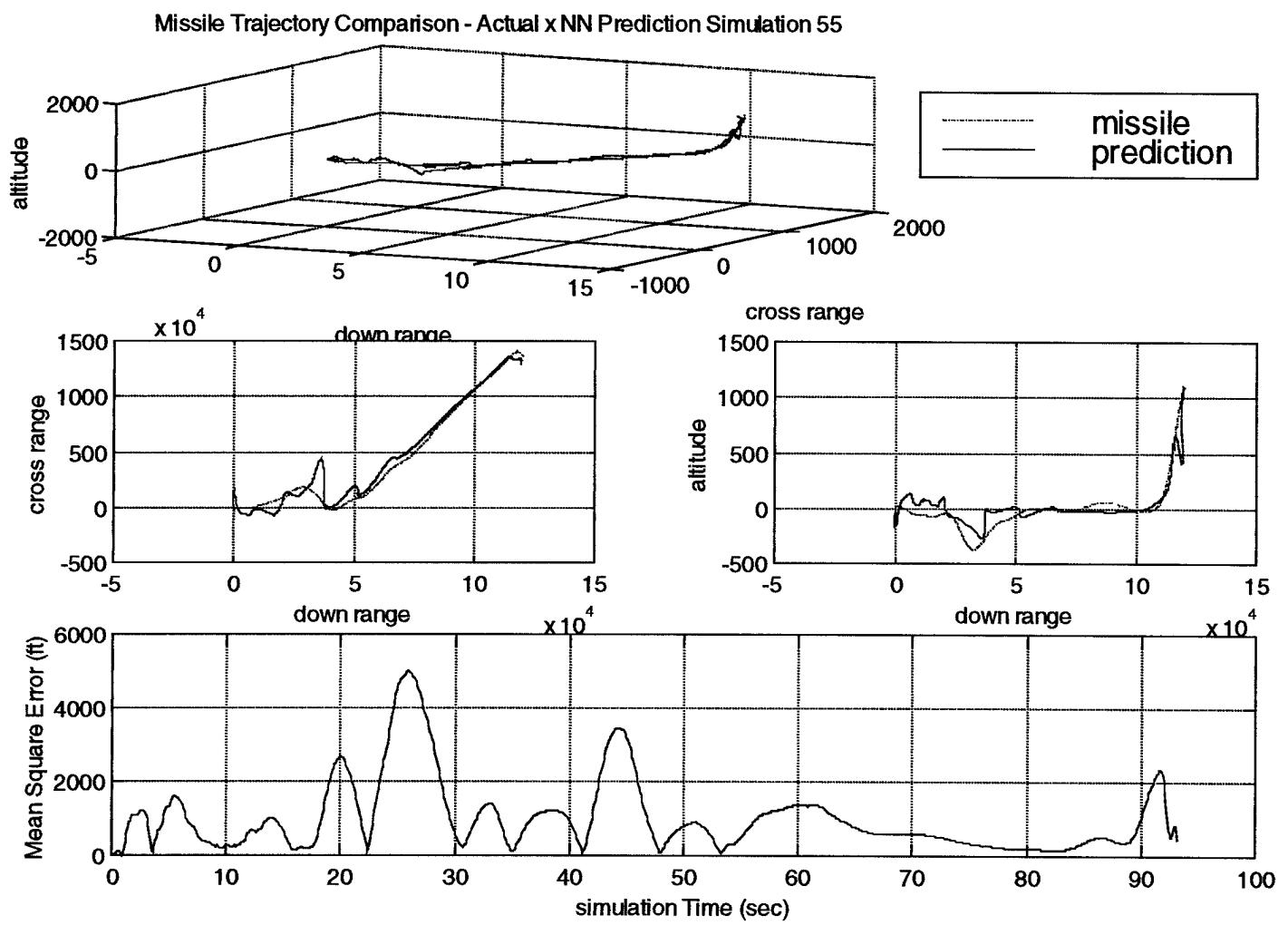


Figure 8.16. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 55

Neural Network Inputs - Simulation 65

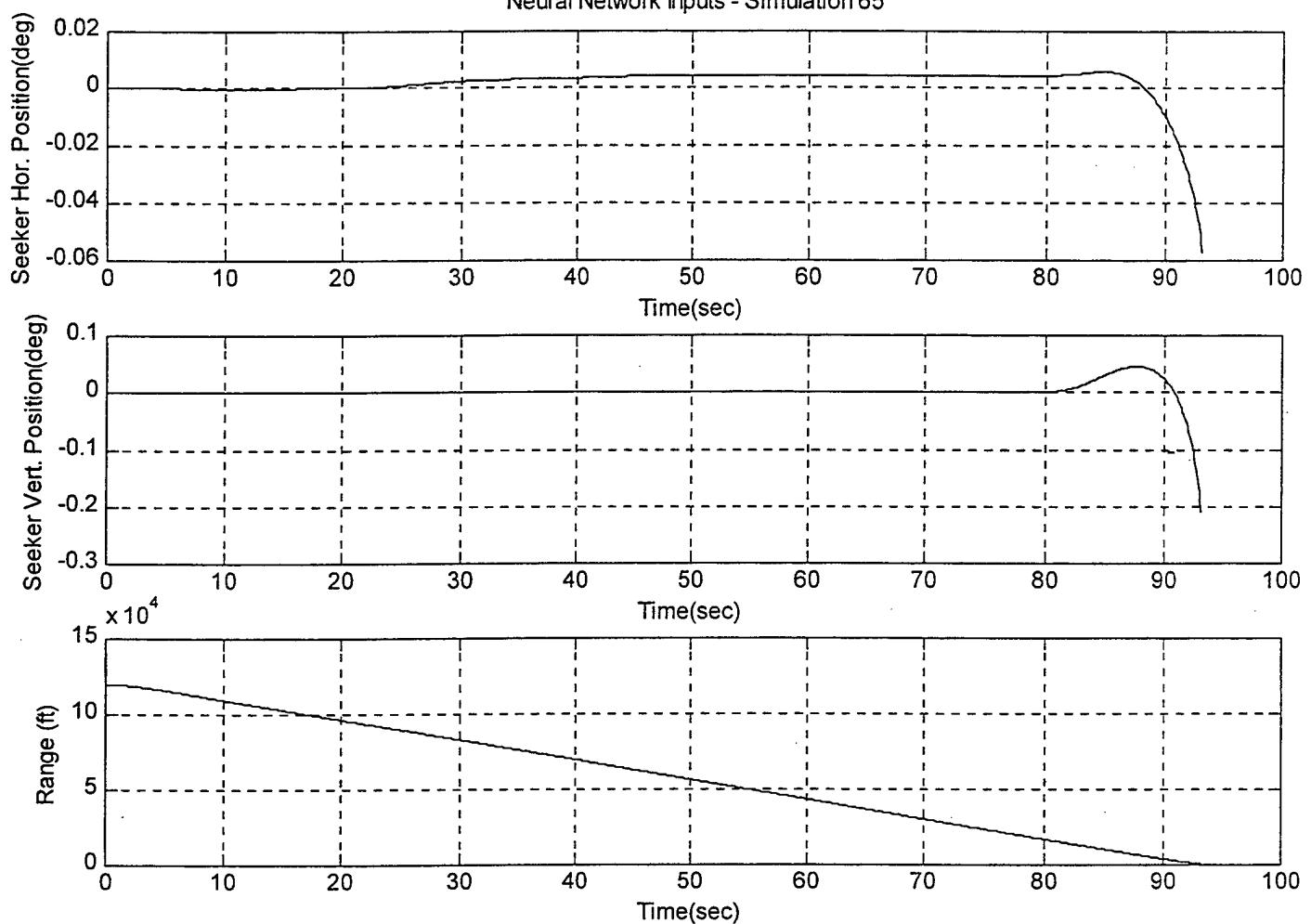


Figure 8.17. Seeker Inputs for the Neural Network (Simulation 65)

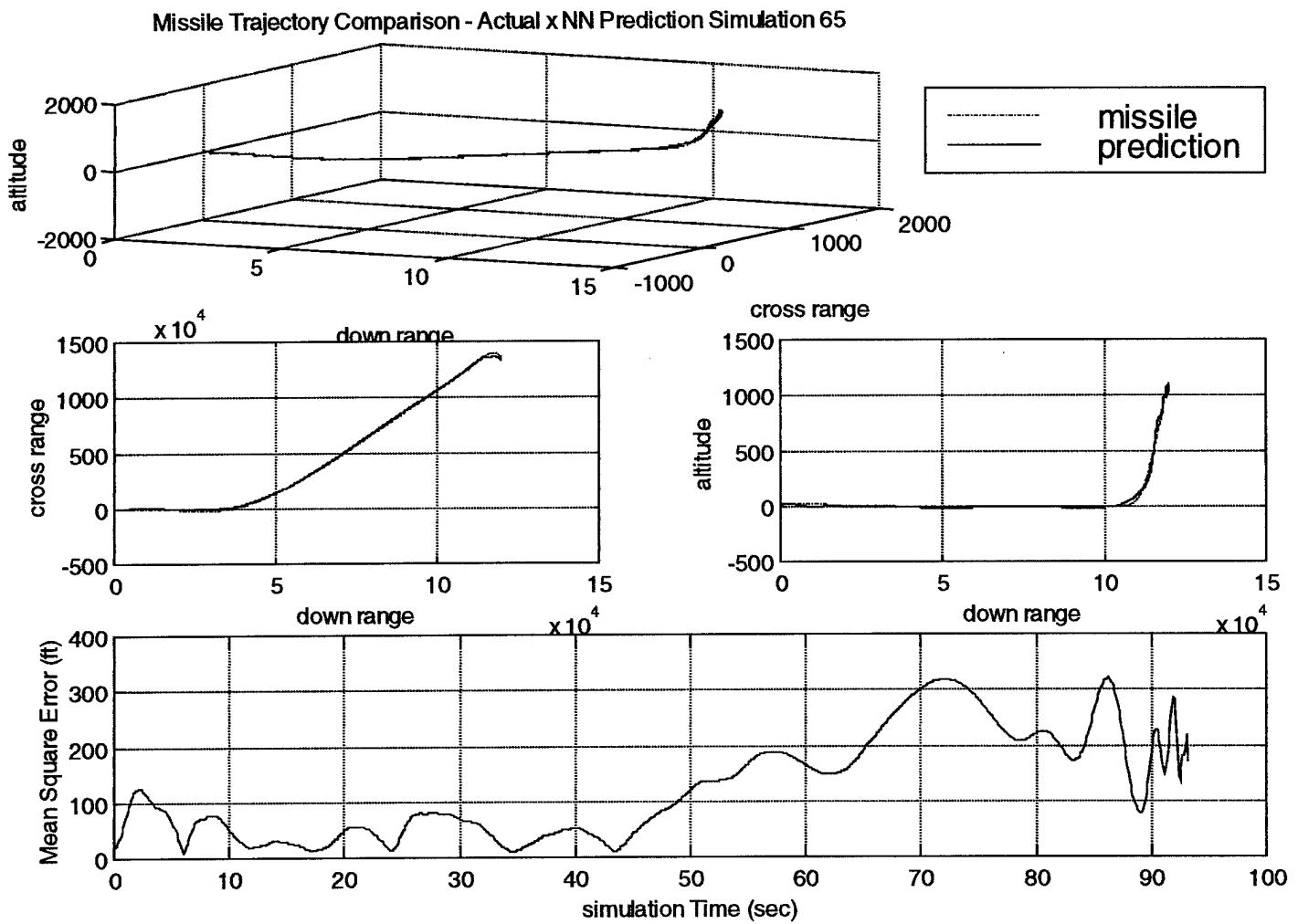


Figure 8.18. Missile Trajectory Comparison – Actual x NN Prediction-Simulation 65

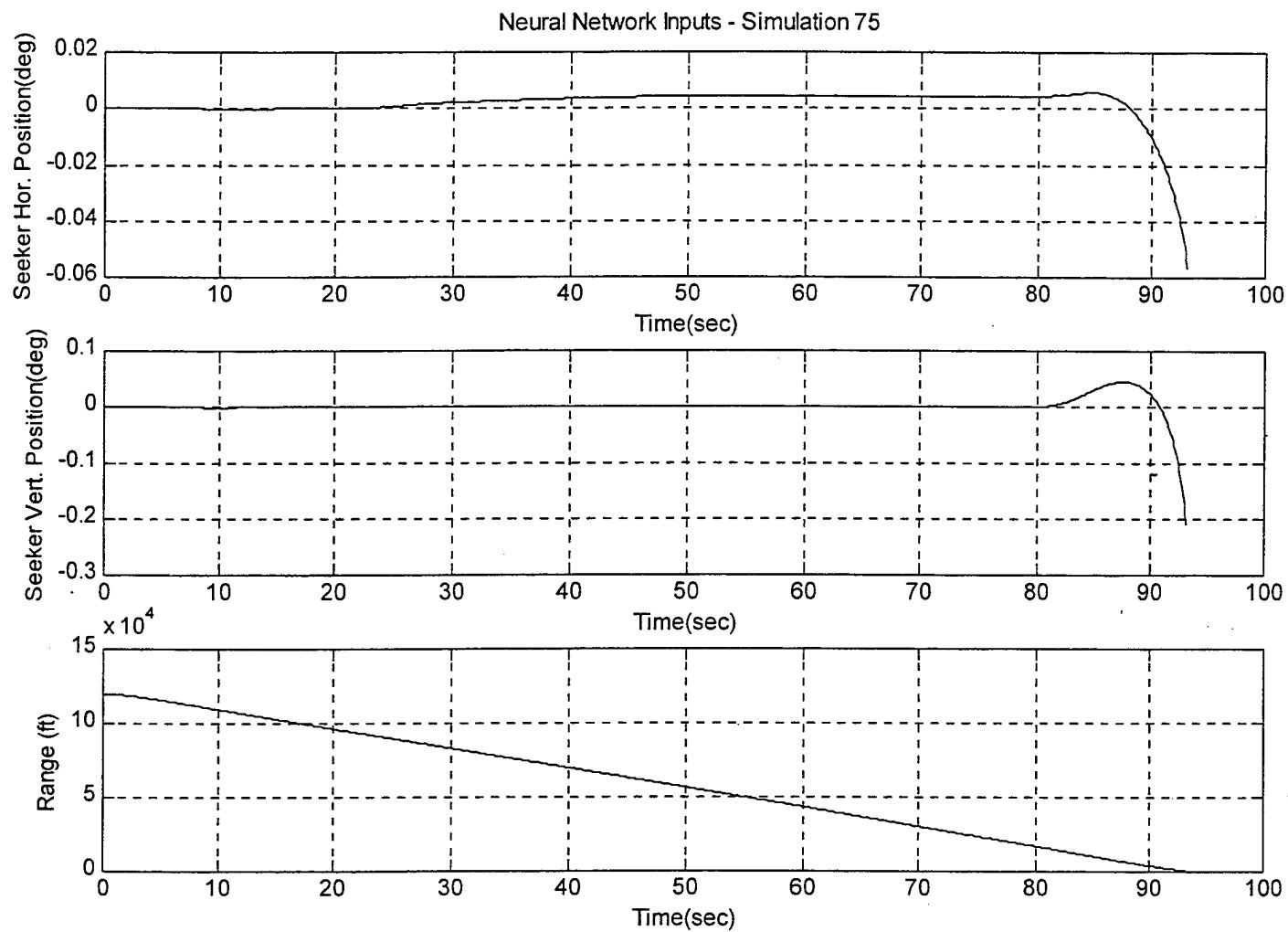


Figure 8.19. Seeker Inputs for the Neural Network (Simulation 75)

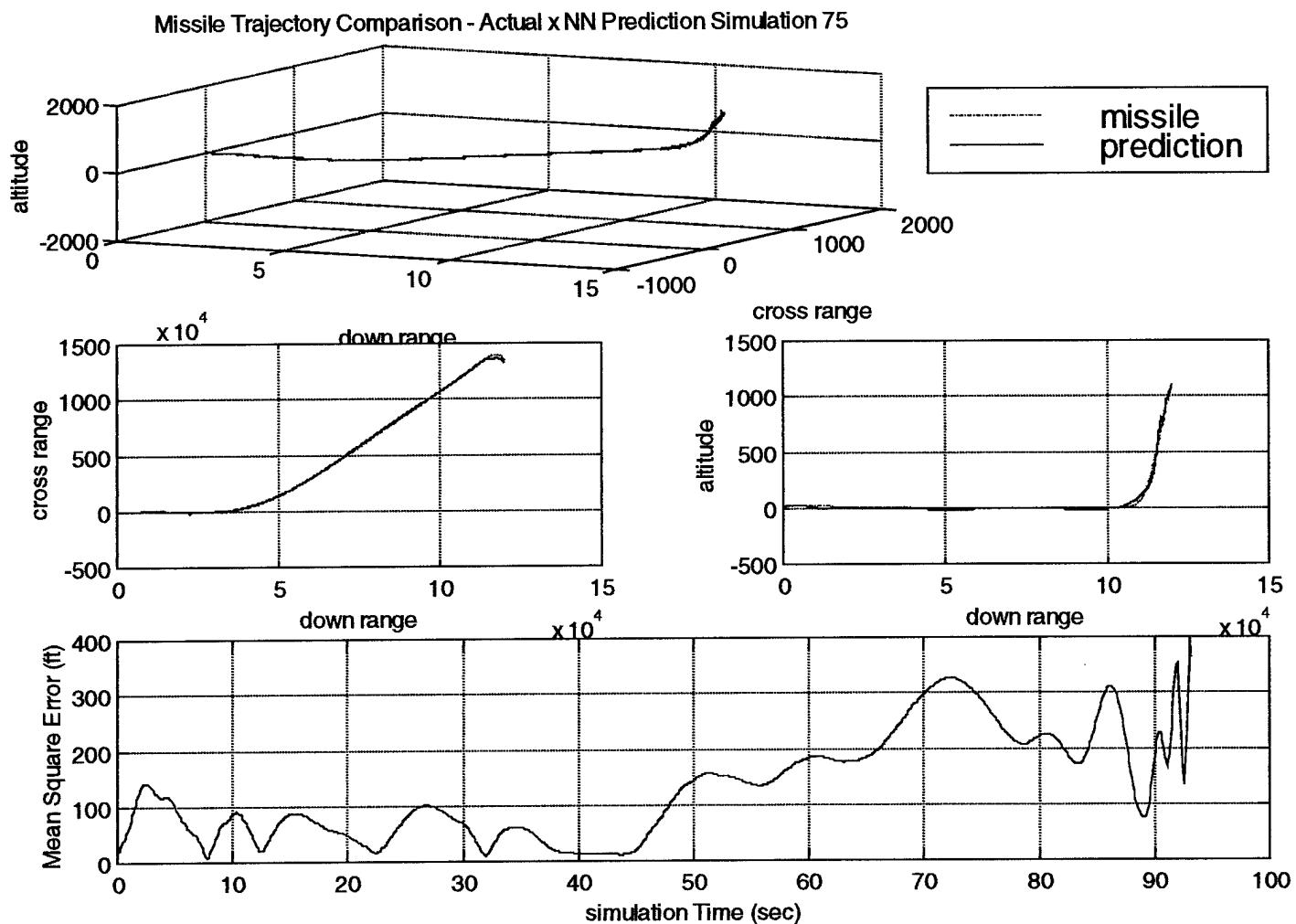


Figure 8.20. Missile Trajectory Comparison – Actual x NN Prediction-Simulation

75

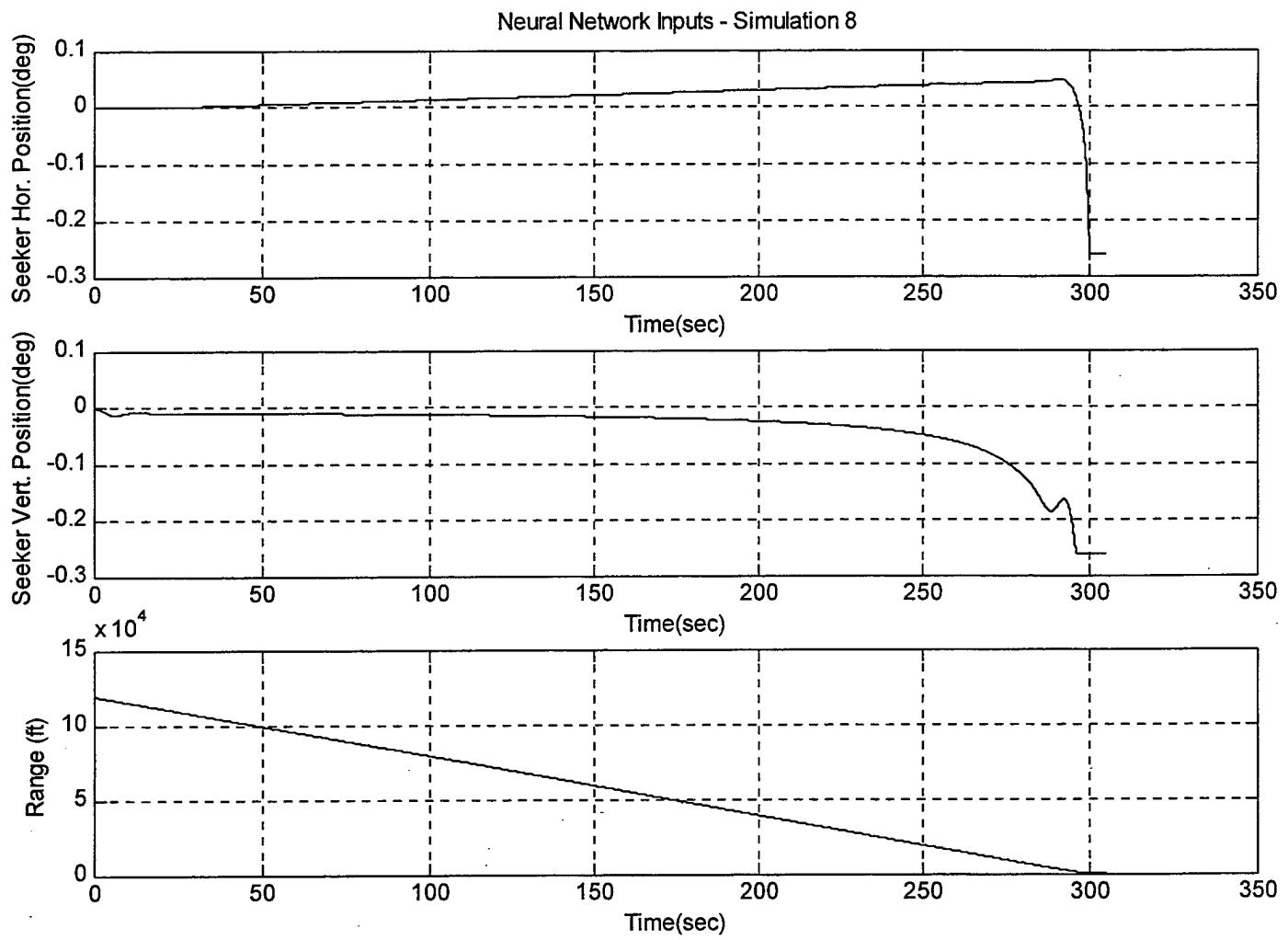


Figure 8.21. Seeker Inputs for the Neural Network (Simulation 8)

Missile/Captive-Carry/Prediction Comparison - Simulation 8

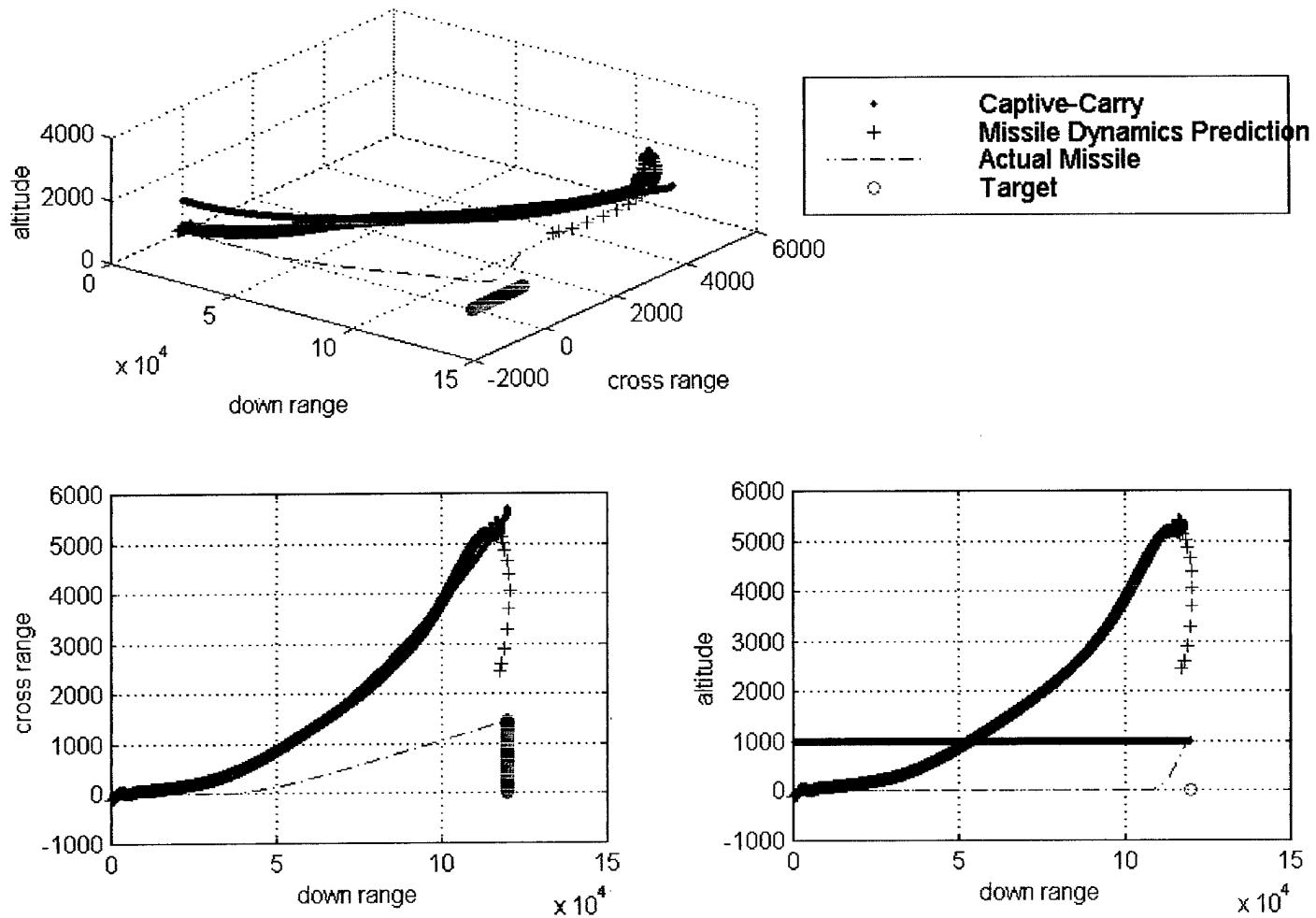


Figure 8.22. Missile/Captive-Carry Prediction Comparison – Simulation 8

IX. CONCLUSIONS

The computer modeling of the captive-carry (open-loop) and missile (closed-loop) simulations using the ASCM Digital Model and the *GUI Menu* allow us to get cost-effective information about the hardware-in-the-loop (HIL) experiments. The model testing also aids in the different approaches for manipulating the data gathered from the experiments in order to predict the electronic attack (EA) measurement of effectiveness from the open-loop simulation.

This thesis produced a comprehensive model description and parameter influence analysis in the missile trajectory. Depending on the information about real ASCMs, the analysis allows us to modify the model parameters or to make model improvements that behave similarly to a real threat. After updating the model for a specific threat, it is possible to study the missile trajectory for the new configuration in order to maximize the effectiveness of the selected EA and, consequently, to increase the miss distance.

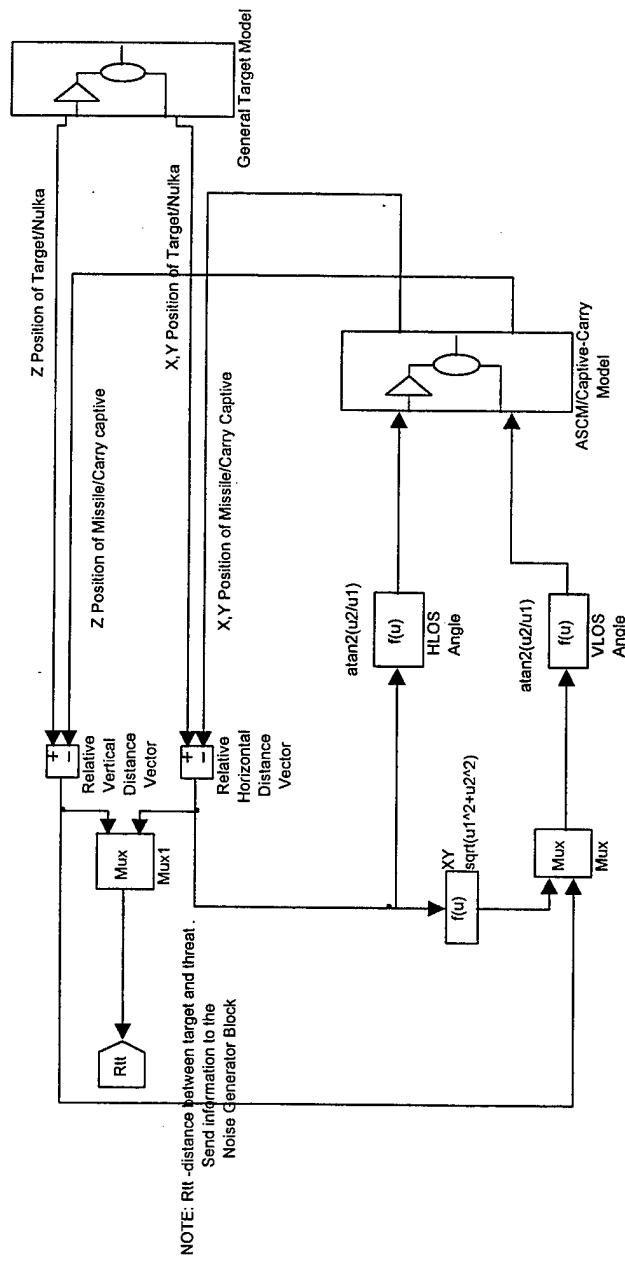
The seeker signal comparisons of the two computer modeling experiments allowed us to conclude that it is feasible to research combining the best results of each computer experiment, i.e., the missile dynamics available in the closed-experiment and the seeker signals in the collision triangle in the open-loop experiment.

Also, in this thesis, it was possible to create an algorithm using a neural network to make a prediction of the missile trajectory and its associated miss distance with and without the presence of the Nulka decoy (EA) using the information of the computer captive-carry experiment.

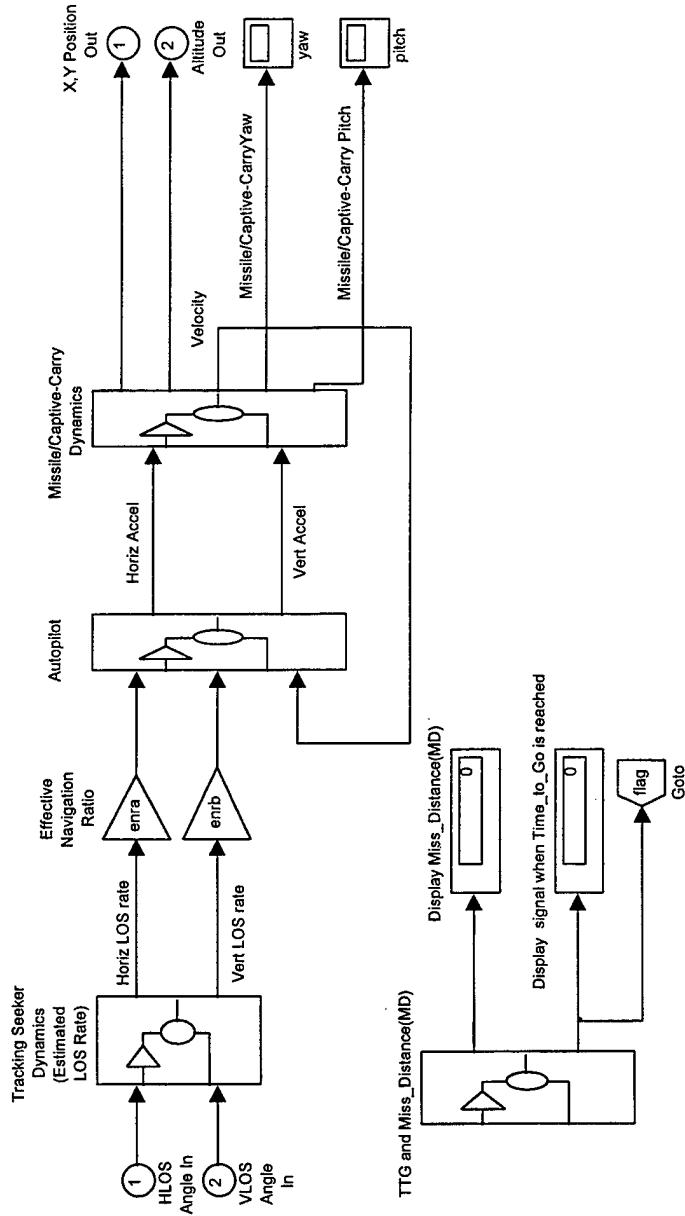
Finally, for further improvements, it is suggested that several target and aircraft profiles (course and velocities) be added in order to observe the results obtained in this thesis in any desired direction. Also, it is suggested that new shipboard countermeasures be included in order to compare the effectiveness of each one using the miss distance parameter. The ASCM model can receive different new sub-systems, for instance, a different seeker model in order to compare its performance with the ones already present in the model, or an autopilot, or new missile dynamics configuration. As said before, the rationale of this computer model is to obtain cost-effective information to improve the already existing HIL technology. The suggestions presented above are made in order to make our computer model more flexible and realistic.

APPENDIX A. ASCM DIGITAL MODEL

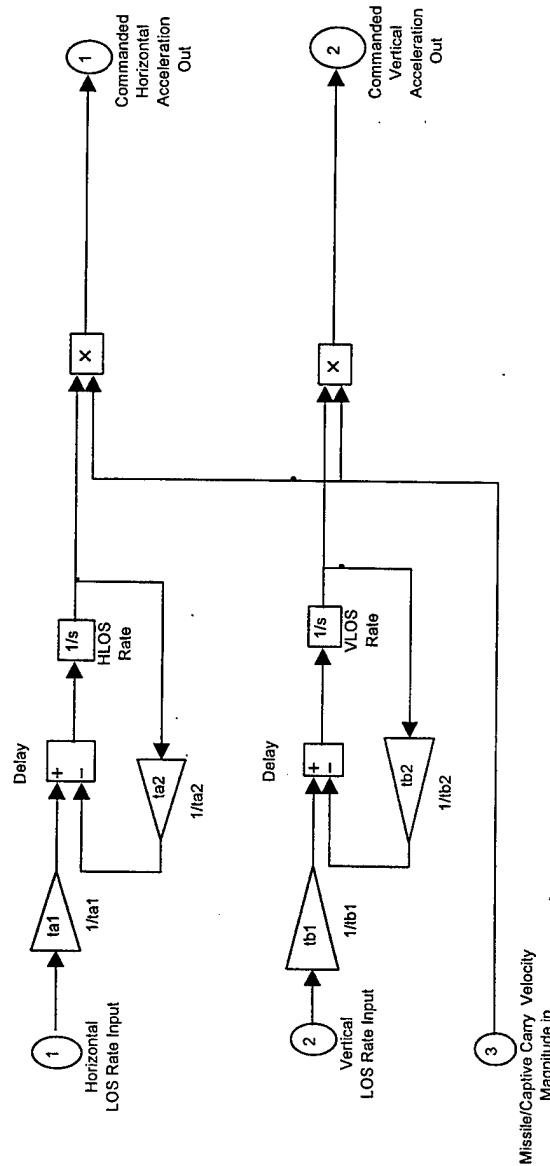
ASCM DIGITAL MODEL



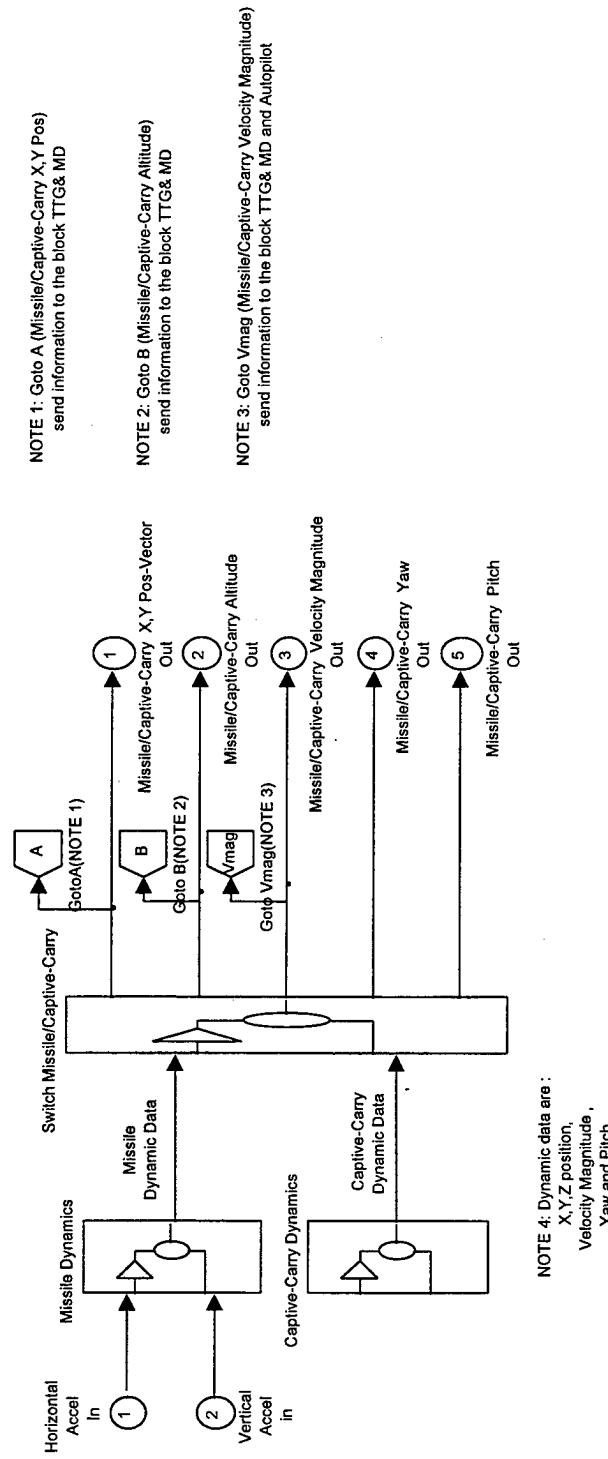
ASCM/Captive-Carry Model



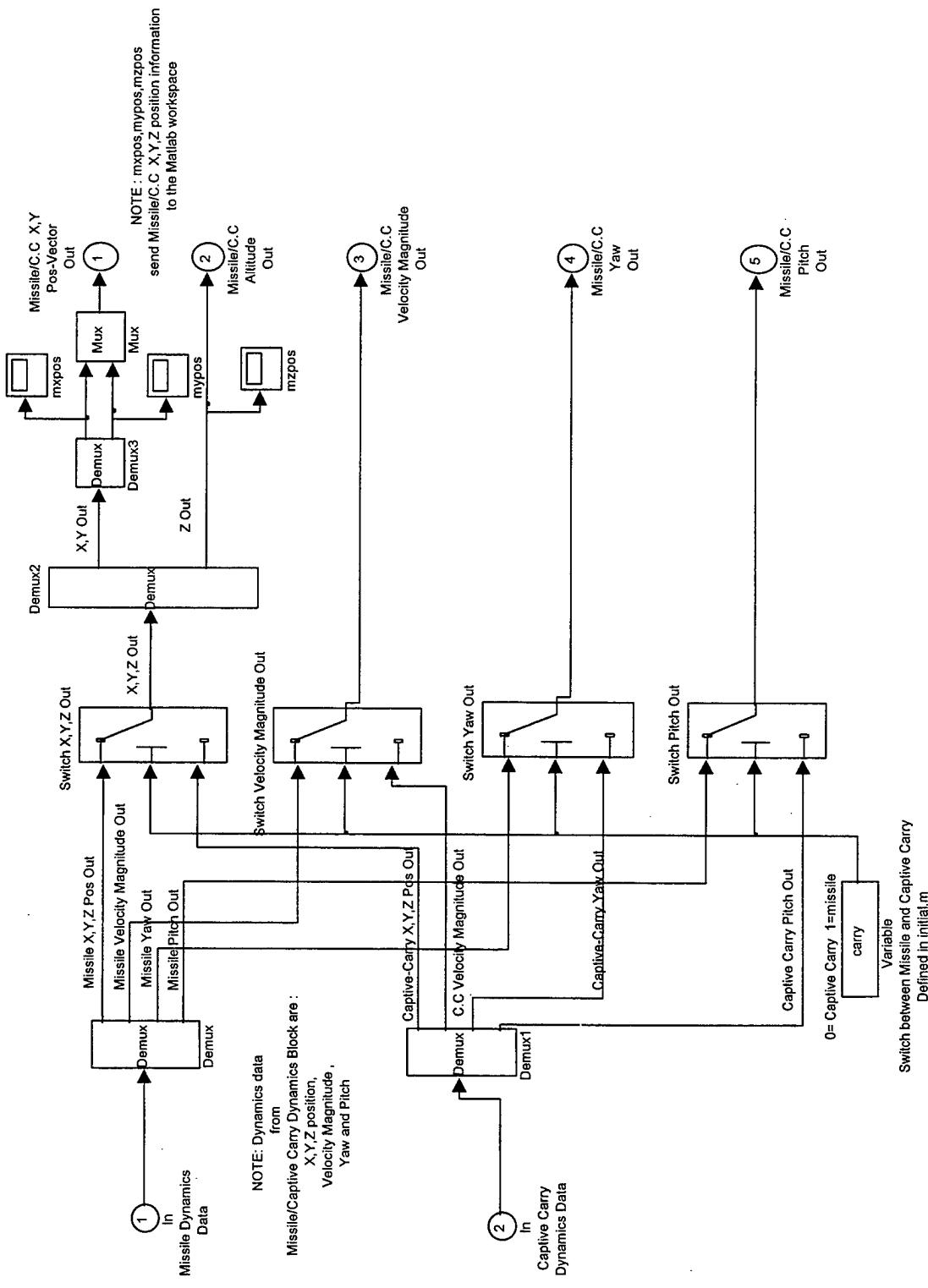
AutoPilot



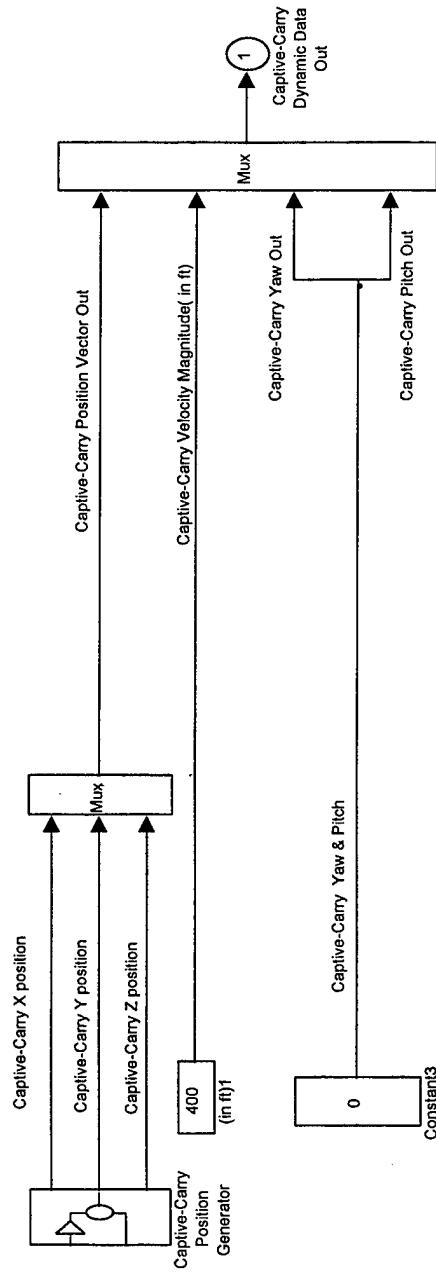
Missile/Captive-Carry Dynamics



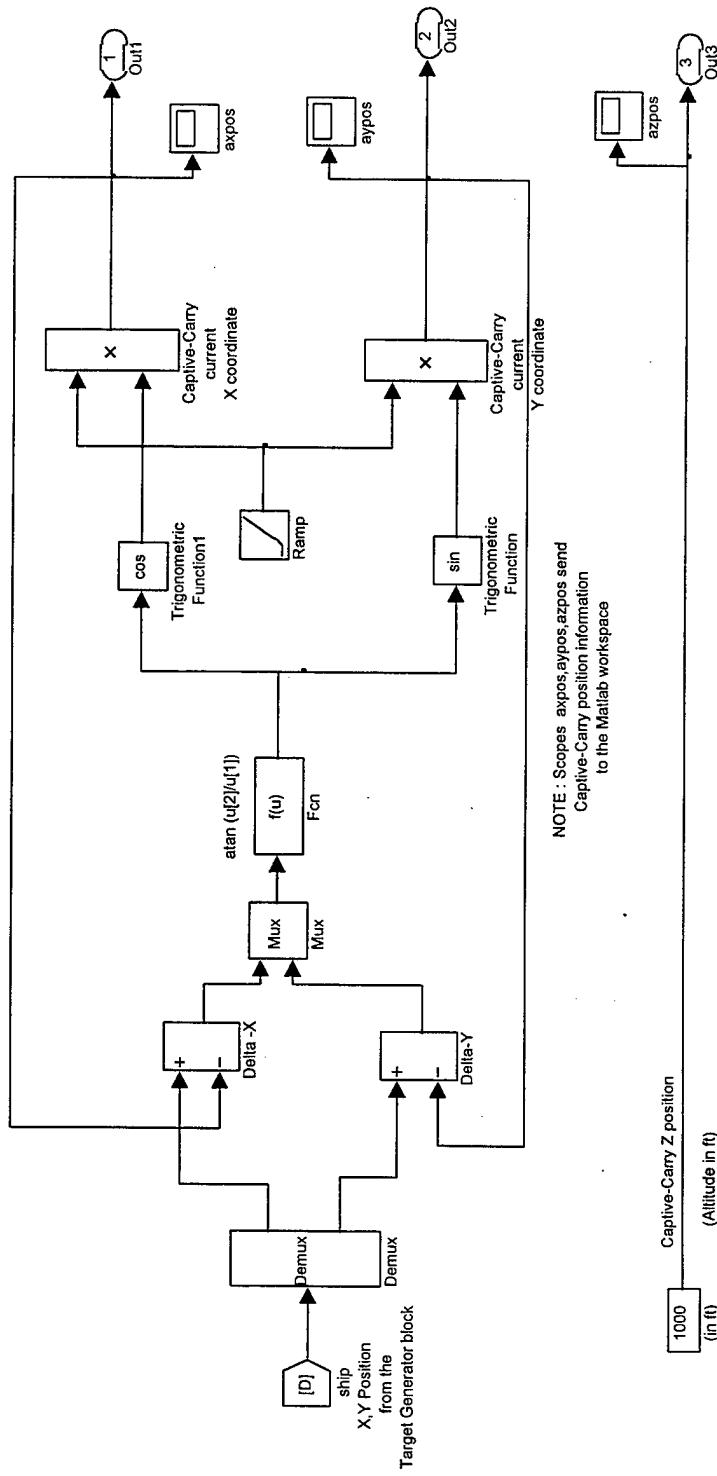
Switch Missile/Captive-Carry



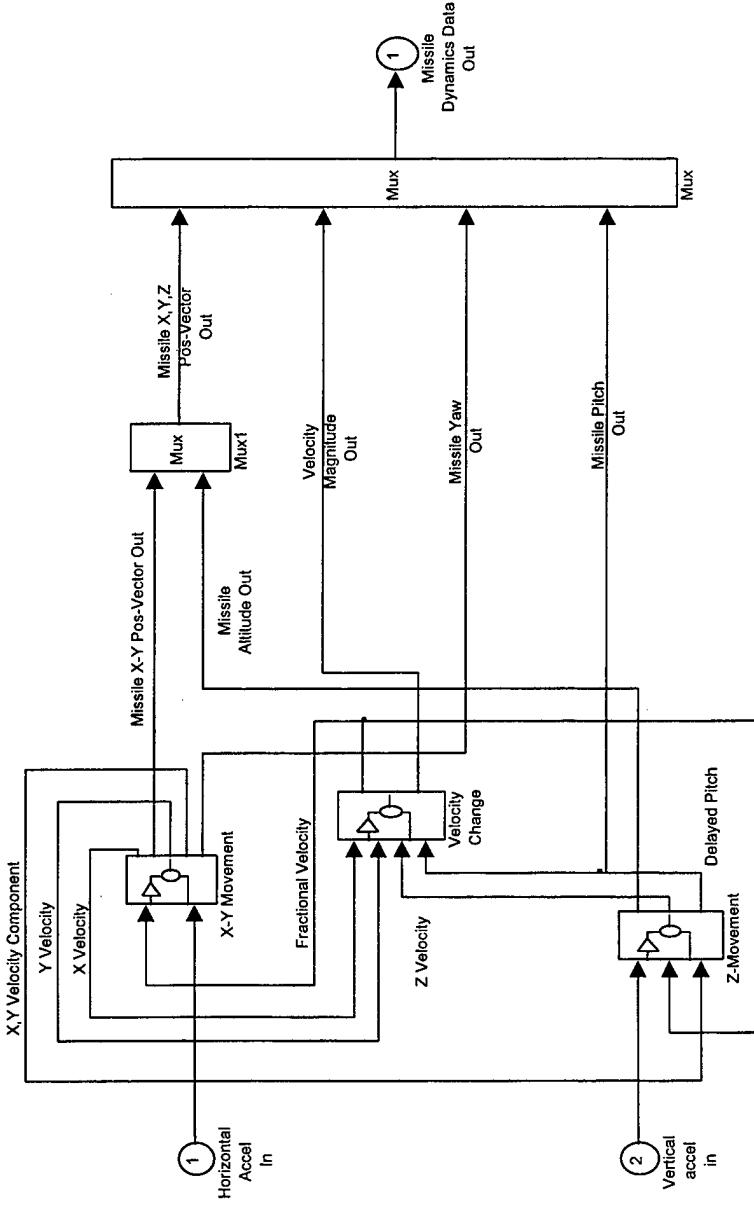
Captive-Carry Dynamics



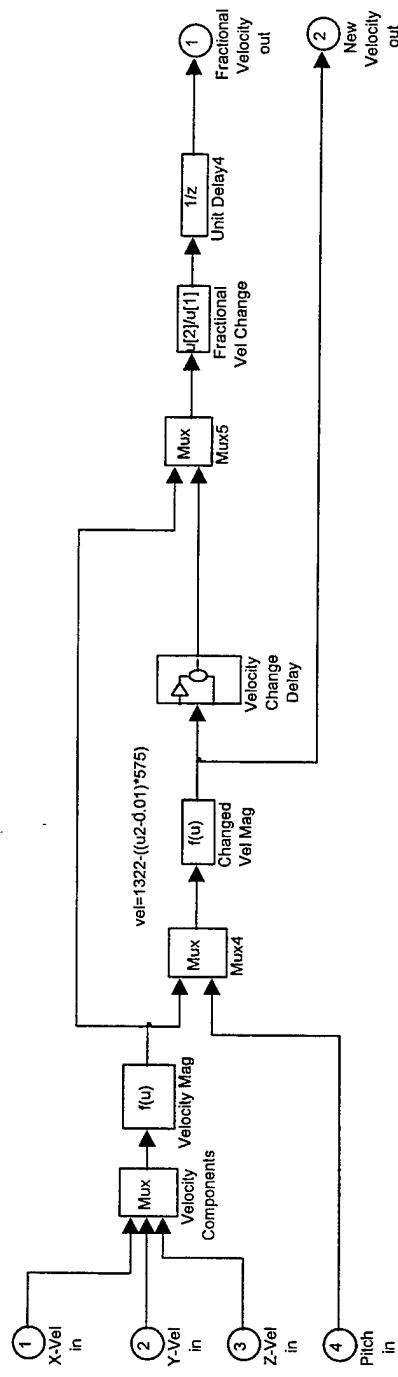
Captive-Carry Position Generator



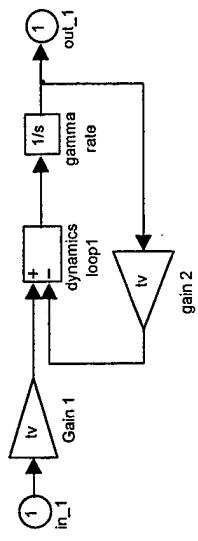
Missile Dynamics



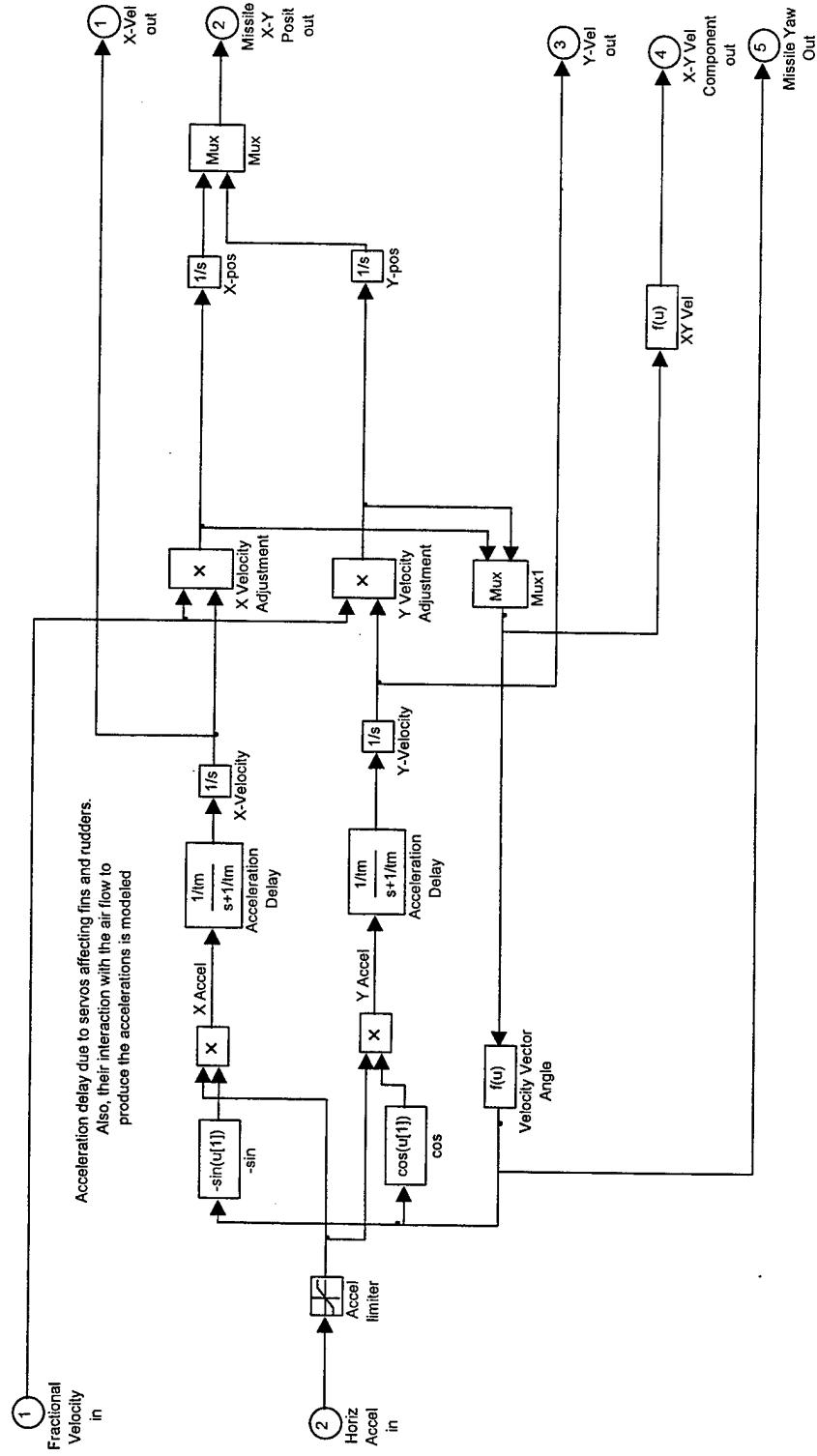
Missile Dynamics
Velocity Change



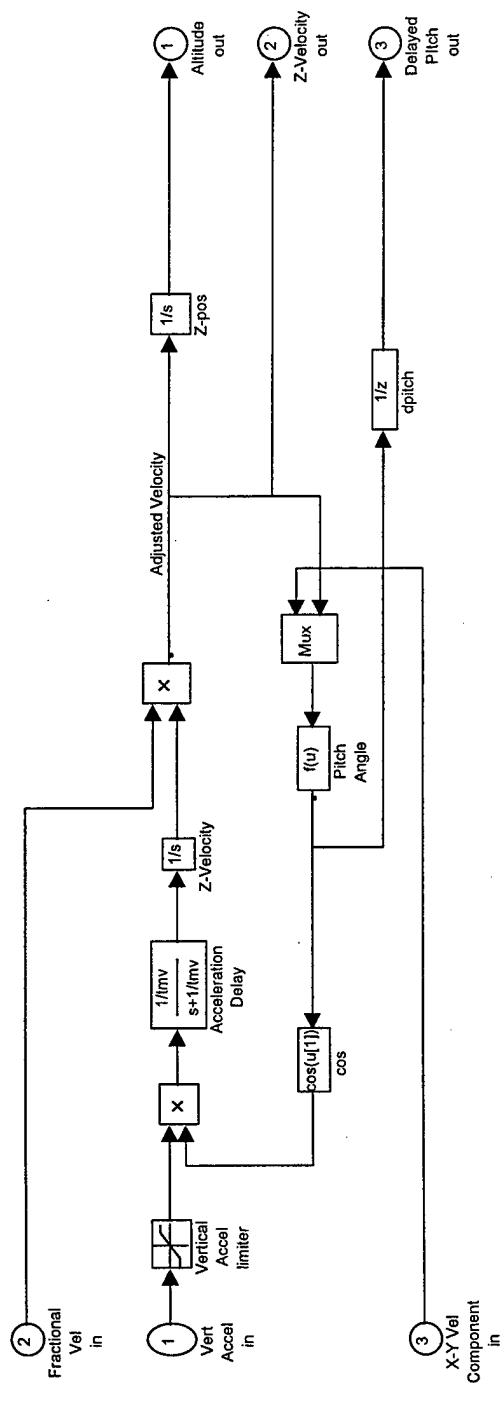
Velocity Change Delay



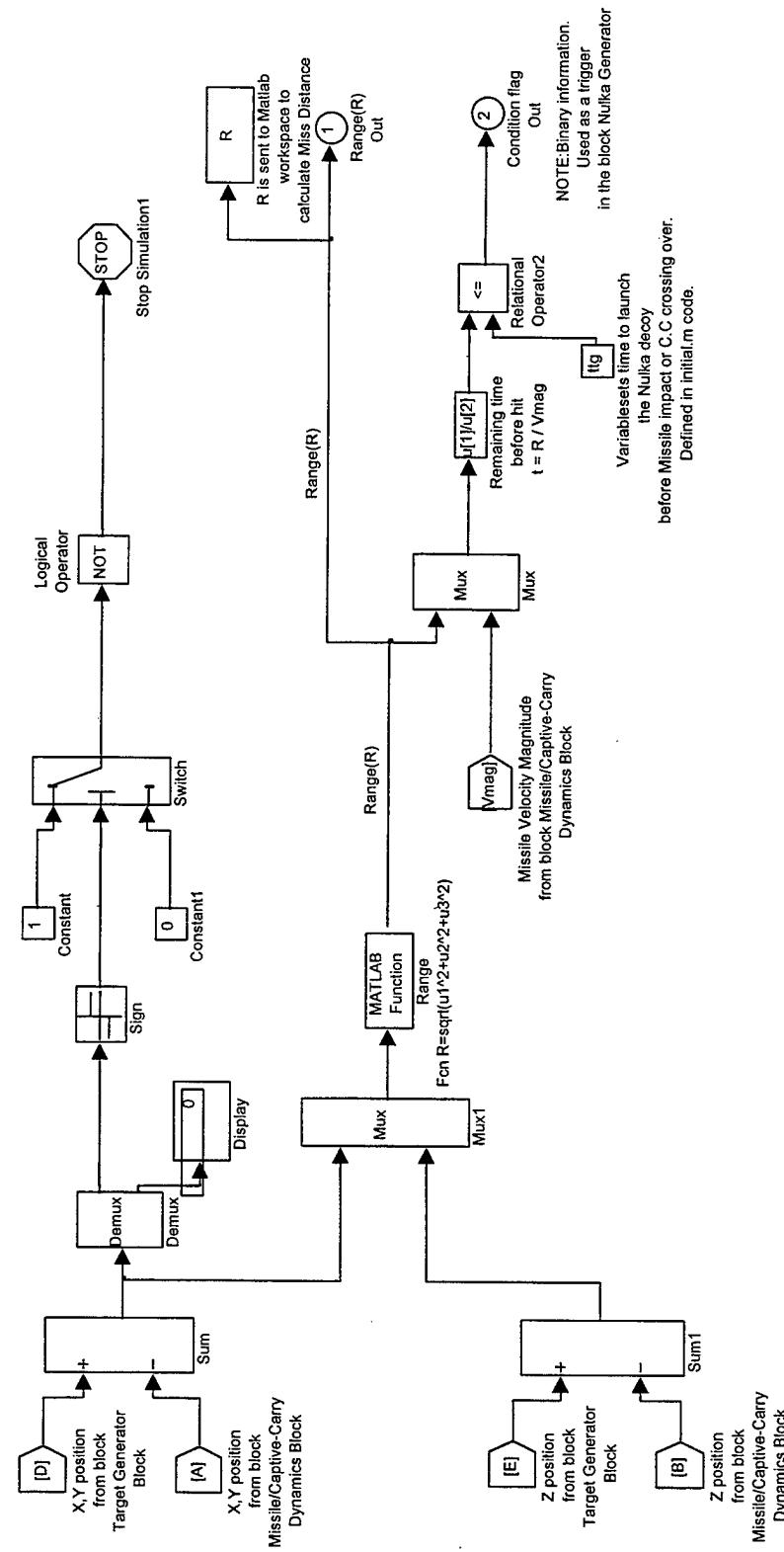
Missile Dynamics X-Y Movement



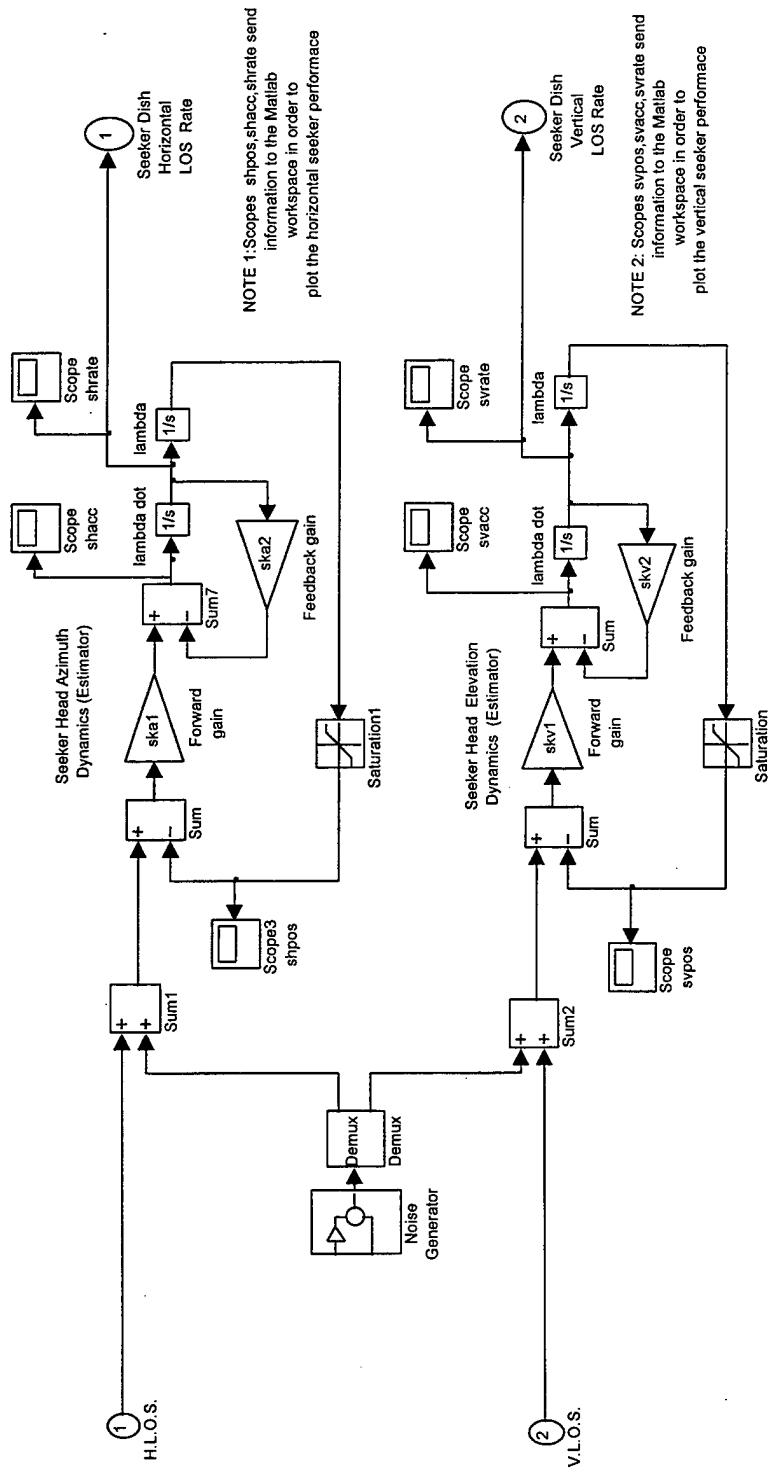
Altitude of Missile Flight Simulation



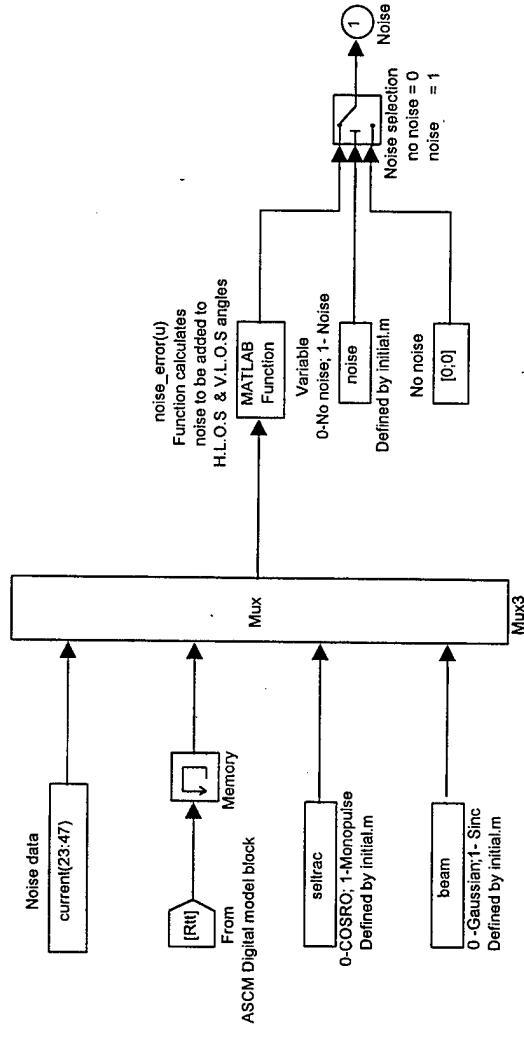
Time_to_Go and Miss_Distance



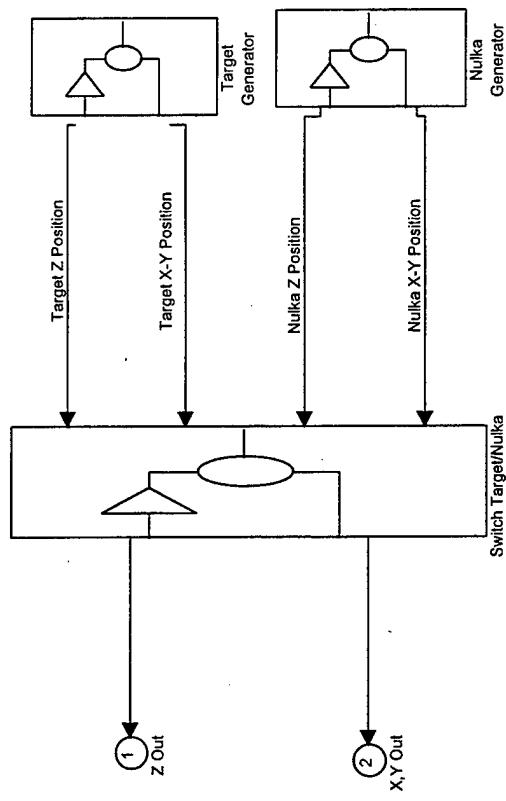
Seeker (Estimatos Rates)



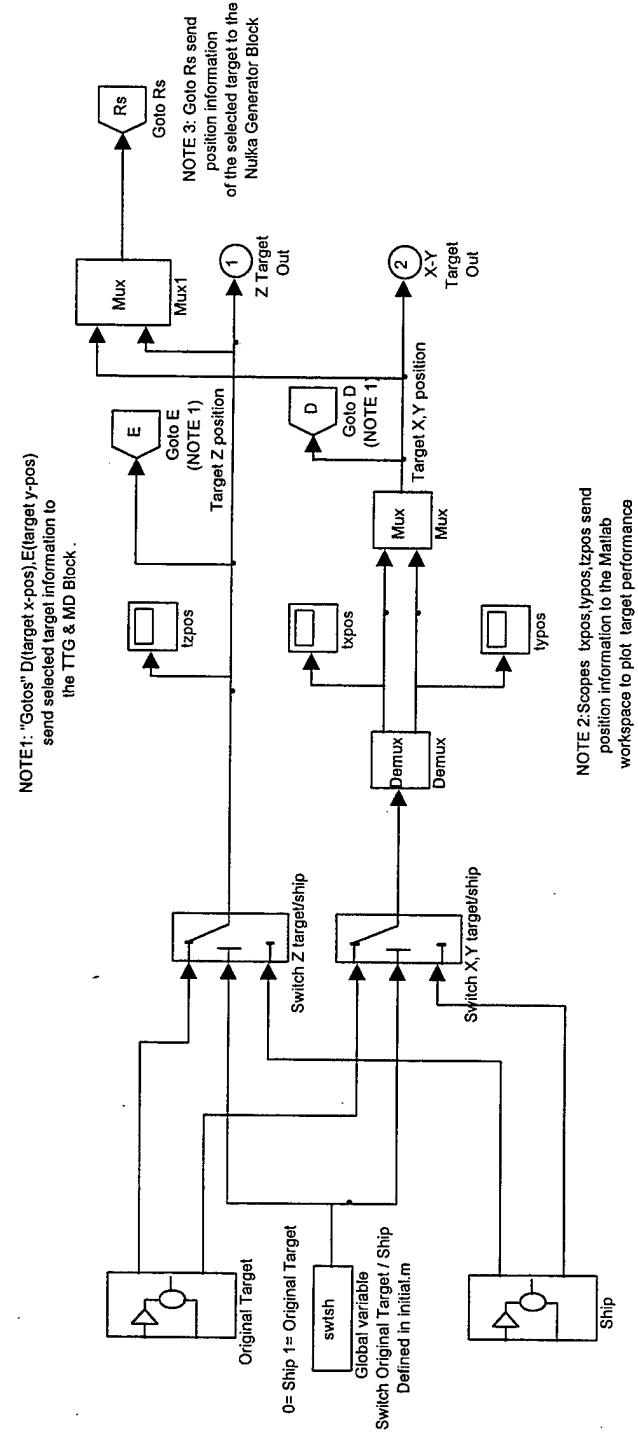
Noise Generator



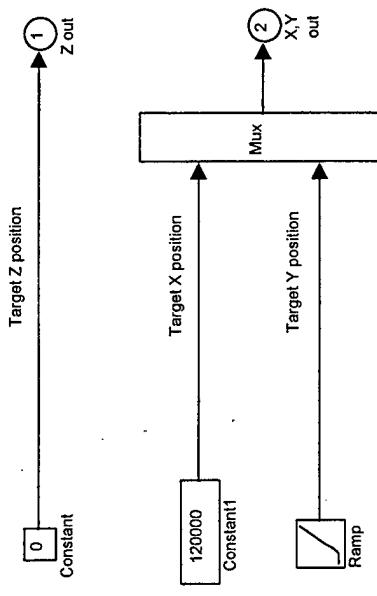
General Target Model



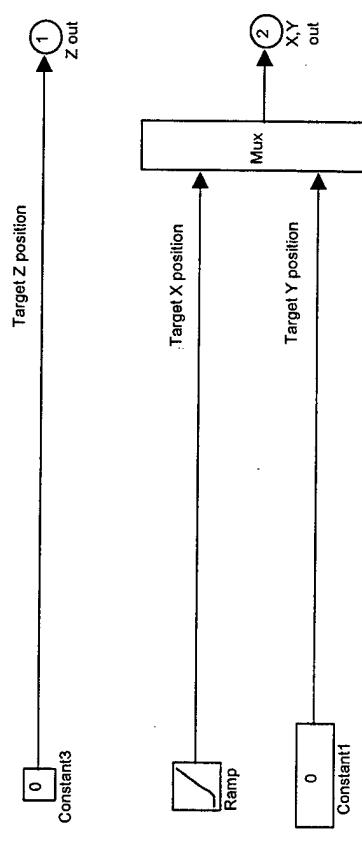
Target Generator

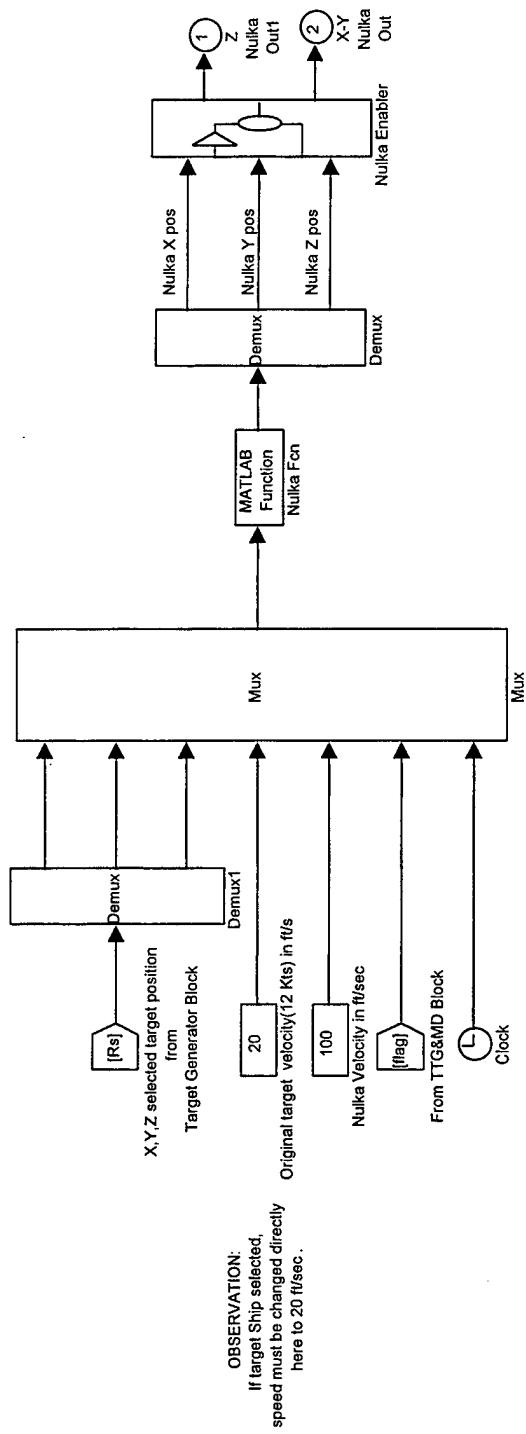


Original Target Generator

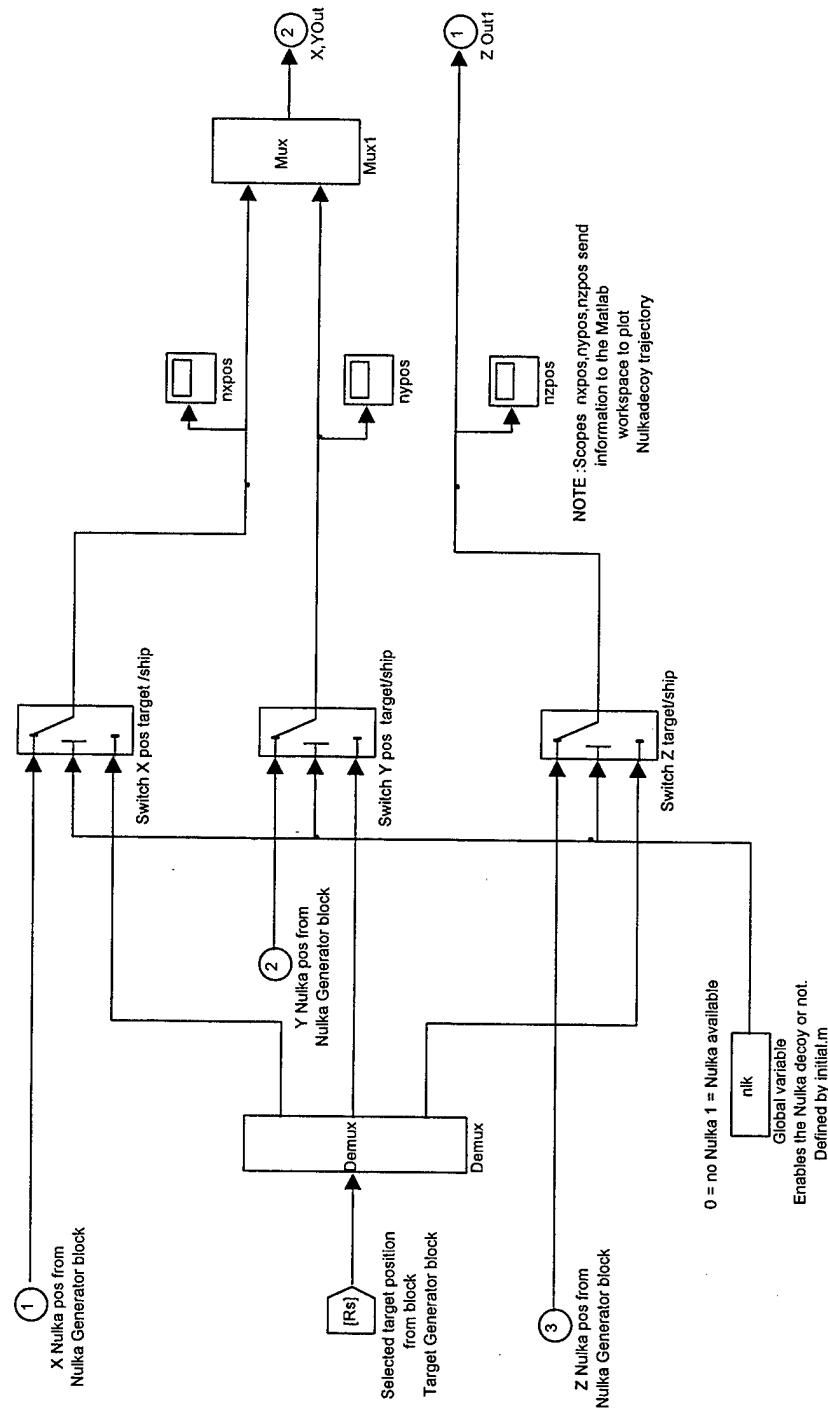


Ship Generator

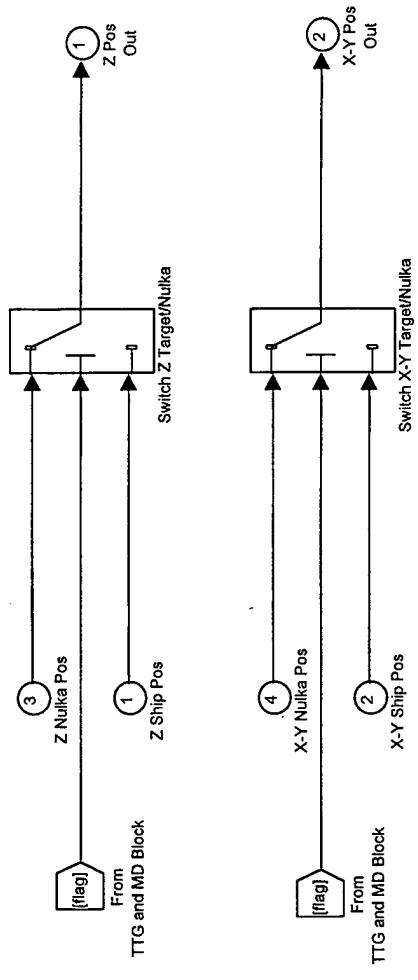




Nukka Enabler



Switch Target/Nukla



APPENDIX B. INITIALIZATION CODE `initial.m`

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: initial.m
%%
%% Type:SCRIPT
%%
%% Purpose: This .m-file stores several values of the
%%           parameters used for
%%           initialization of the ASCM Digital Model as well
%%           as the values of the seeker
%%           parameters used for noise calculation by the -
%%           function noise.m.
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
%% clear all
%% Clear workspace to run initial.m code

global po I % po and I are global variables
% po is a matrix used for
% controlling the Nulka Rocket
% launching
% I is a structure that stores the
% initialization
% parameters used by ASCM Digital
% Model and noise calculation

%=====
%
% Define initialization structure I
%
% General Switches

% num=simulation number (referred to the missile simulation)
% swtsh=switch_to_ship
% carry=threat selection

```

```
% nlk=Nulka enabler  
% chaff= chaff enabler  
% noise= noise enabler  
% seltrac = Select type of tracking ( 0 = COSRO ,  
% 1 =MONOPULSE )
```

```
%  
% ASCM Digital Model
```

```
% skal=seeker foward gain(azimuth)  
% ska2=seeker feedback gain(azimuth)  
% skv1=seeker foward gain(vertical)  
% skv2=seeker feedback gain(vertical)  
% enra=effective navigational ratio (foward gain-azimuth)  
% enrb=effective navigational ratio (foward gain-vertical)  
% ta1=autopilot foward gain(azimuth)  
% ta2=autopilot feedback gain(azimuth)  
% tb1=autopilot foward gain(vertical)  
% tb2=autopilot feedback gain(vertical)  
% tm=acceleration delay(X-Y movement)  
% tmv=acceleration delay(Z movement)  
% tv=velocity change delay  
% ttg=time_to_go  
% sekhlim=seeker horizontal limit(15 degees)  
% sekvlim=seeker vertical limit(15 degees)
```

```
%  
% COSRO SEEKER
```

```
% ppc= Peak Power (kW)  
% freqc= Frequency (GHz)  
% antgainc= Antenna Gain (dB)  
% Hpc = Half Power Beam Width-3 dB (degrees)  
% noibwc = Noise Bandwidth(MHz)  
% noifigc = Noise Figure (dB)  
% rrc = Range Resolution (m)  
% numpulc = Number of pulses integrated  
% nosangc = Normalized offset angle  
% envc = Envelope Correlation Time (s)  
% nutfreqc = Nutation Frequency (Hz)  
% serbwbc = Servo Bandwidth (Hz)  
% crossc = Target Radar Cross Section (m^2)  
% alphac = One Way Atmospheric Attenuation(dB/Km)  
% beam = Beam Shape ( 0 = gaussian , 1 = sinc )
```

```
%  
% MONOPULSE SEEKER
```

```
% ppm= Peak Power (kW)  
% freqm= Frequency (GHz)  
% antgainm= Antenna Gain (dB)  
% Hpm = Half Power Beam Width-3 dB (degrees)
```

```

% noibwm = Noise Bandwidth (MHz)
% noifigm = Noise Figure (dB)
% rrm = Range Resolution (m)
% numpulm = Number of pulses integrated
% fsrlm = First Side Lobe Ratio (dB)
% crossm = Target Radar Cross Section (m^2)
% alphac = One Way Atmospheric Attenuation ( dB/Km)

%=====
%
% Create cells that will be used to store the initialization
% parameters. The cells are
% required for dealing with the Matlab structure.

A=cell(1,1);
B=cell(1,1);
C=cell(1,1);
D=cell(1,1);
E=cell(1,1);
F=cell(1,1);
G=cell(1,1);
H=cell(1,1);
J=cell(1,1);
K=cell(1,1);
L=cell(1,1);
M=cell(1,1);
N=cell(1,1);
O=cell(1,1);
P=cell(1,1);
Q=cell(1,1);
R=cell(1,1);
S=cell(1,1);
T=cell(1,1);
U=cell(1,1);
V=cell(1,1);
W=cell(1,1);
X=cell(1,1);
Y=cell(1,1);
Z=cell(1,1);
a=cell(1,1);
b=cell(1,1);
c=cell(1,1);
d=cell(1,1);
e=cell(1,1);
f=cell(1,1);
g=cell(1,1);
h=cell(1,1);
k=cell(1,1);
l=cell(1,1);

```

```

m=cell(1,1);
n=cell(1,1);
o=cell(1,1);
p=cell(1,1);
q=cell(1,1);
r=cell(1,1);
s=cell(1,1);
t=cell(1,1);
u=cell(1,1);
v=cell(1,1);
w=cell(1,1);
x=cell(1,1);
y=cell(1,1);
z=cell(1,1);

%=====
%
% Create values that will be assigned to cells created
% previously.
% These values are easier to read in the Appendix L in an
% Excel database format
%
aux1=[zeros(1,10) ,ones(1,10),ones(1,10),ones(1,10)];
aux2=[aux1(ones(1,60),:) ]';
aux3=[zeros(1,30),ones(1,10)] ;
aux4=[aux3(ones(1,60),:) ]';
aux5=[zeros(1,20),ones(1,10),zeros(1,10)] ;
aux6=[aux5(ones(1,60),:) ]';
aux7=[2 4 6 8 10 12 14 16 18 20];
aux8=[aux7(ones(1,240),:) ]';

b1=0.25;b2=0.1;b3=1;b4=4;

aux9=[[b1(ones(1,40),:) ] [b2(ones(1,40),:) ]
[b3(ones(1,40),:) ] [b4(ones(1,40),:) ]];
aux10=[[b1(ones(1,40),:) ] [b1(ones(1,40),:) ] [b1(ones(1,40),:)
] [b1(ones(1,40),:) ] ...
[b1(ones(1,40),:) ] [b2(ones(1,40),:) ]
[b3(ones(1,40),:) ] [b4(ones(1,40),:) ]];
aux11=[[b1(ones(1,40),:) ] [b1(ones(1,40),:) ] [b1(ones(1,40),:)
] [b1(ones(1,40),:) ]...
[b1(ones(1,40),:) ] [b1(ones(1,40),:) ] [b1(ones(1,40),:)
] [b1(ones(1,40),:) ]...
[b1(ones(1,40),:) ] [b2(ones(1,40),:) ]
[b3(ones(1,40),:) ] [b4(ones(1,40),:) ]];
aux12=[[b1(ones(1,40),:) ] [b1(ones(1,40),:) ] [b1(ones(1,40),:)
] [b1(ones(1,40),:) ]...

```

```

[b1(ones(1,40),:)] [b1(ones(1,40),:)] [b1(ones(1,40),:)] [b1(on
es(1,40),:)];

[b1(ones(1,40),:)] [b1(ones(1,40),:)] [b1(ones(1,40),:)] [b1(on
es(1,40),:)];

[b1(ones(1,40),:)] [b2(ones(1,40),:)];
[b3(ones(1,40),:)] [b4(ones(1,40),:)];

b5=1;b6=2;b7=4;b8=8;

aux13=[[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)]
[b7(ones(1,40),:)];

[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(on
es(1,40),:)];

[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(on
es(1,40),:)];

[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(on
es(1,40),:)];

[b5(ones(1,40),:)] [b6(ones(1,40),:)];
[b7(ones(1,40),:)] [b8(ones(1,40),:)];

aux14=[[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)]
[b7(ones(1,40),:)];

[b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(ones(1,40),:)] [b7(on
es(1,40),:)];

[b5(ones(1,40),:)] [b6(ones(1,40),:)];
[b7(ones(1,40),:)] [b8(ones(1,40),:)];

b9=0.25;b10=0.5;b11=1;b12=4;

aux15=[[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40)
,:)] [b10(ones(1,40),:)];

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)];

```



```

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...
    [b9(ones(1,40),:)] [b10(ones(1,40),:)]
[b11(ones(1,40),:)] [b12(ones(1,40),:)]];
aux18=[[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40)
,:)] [b10(ones(1,40),:)] ...

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...
    [b9(ones(1,40),:)] [b10(ones(1,40),:)]
[b11(ones(1,40),:)] [b12(ones(1,40),:)]];

b13=0.5;b14=1;b15=2;b16=4;

aux19=[[b15(ones(1,40),:)] [b15(ones(1,40),:)] [b15(ones(1,40)
,:)] [b15(ones(1,40),:)] ...

```



```

[b15(ones(1,40),:)] [b15(ones(1,40),:)] [b15(ones(1,40),:)] [b1
5(ones(1,40),:)] ...

[b15(ones(1,40),:)] [b15(ones(1,40),:)] [b15(ones(1,40),:)] [b1
5(ones(1,40),:)] ...

[b15(ones(1,40),:)] [b15(ones(1,40),:)] [b15(ones(1,40),:)] [b1
5(ones(1,40),:)] ...

[b15(ones(1,40),:)] [b15(ones(1,40),:)] [b15(ones(1,40),:)] [b1
5(ones(1,40),:)] ...
    [b13(ones(1,40),:)] [b14(ones(1,40),:)]
[b15(ones(1,40),:)] [b16(ones(1,40),:)];
aux21=[[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40)
,:)] [b10(ones(1,40),:)] ...

[b10(ones(1,40),:)] [b10(ones(1,40),:)] [b10(ones(1,40),:)] [b1
0(ones(1,40),:)] ...

```

```

[b9(ones(1,40),:)] [b10(ones(1,40),:)];
[b11(ones(1,40),:)] [b12(ones(1,40),:)];

b17=0.1309;b18=0.2618;b19=0.3491;b20=0.5236;

aux22=[[b18(ones(1,40),:)] [b18(ones(1,40),:)] [b18(ones(1,40),
,:)] [b18(ones(1,40),:)] ...

[b18(ones(1,40),:)] [b18(ones(1,40),:)] [b18(ones(1,40),:)] [b1
8(ones(1,40),:)] ...

[b17(ones(1,40),:)] [b18(ones(1,40),:)]
[b19(ones(1,40),:)] [b20(ones(1,40),:)]];
aux23=[[b18(ones(1,40),:)] [b18(ones(1,40),:)] [b18(ones(1,40)
,:)] [b18(ones(1,40),:)] ...

```



```

a20=[10];
a22=[11];
a24=[100];
a26=[0.06];
a28=[60];
a30=[3000];
a32=[0.055];
a34=[24];
a40=[30];
%
%=====
%
% Assign the values of the initialization parameters to the
% cells.
%
A={[1:1:2400]'}};
B={[ones(1,2400)']'};
C={[ones(1,2400)']'};
D={[ones(1,2400)']'};
E={[zeros(1,2400)']'};
F={aux2(:)};
G={aux4(:)};
H={[aux9(:); b1(ones(2240,1),:)]};
J={[aux10(:); b1(ones(2080,1),:)]};
K={[aux11(:); b1(ones(1920,1),:)]};
L={[aux12(:); b1(ones(1760,1),:)]};
M={[aux13(:); b7(ones(1600,1),:)]};
N={[aux14(:); b7(ones(1440,1),:)]};
O={[aux15(:); b10(ones(1280,1),:)]};
P={[aux16(:); b10(ones(1120,1),:)]};
Q={[aux17(:); b10(ones(960,1),:)]};
R={[aux18(:); b10(ones(800,1),:)]};
S={[aux19(:); b15(ones(640,1),:)]};
T={[aux20(:); b15(ones(480,1),:)]};
U={[aux21(:); b10(ones(320,1),:)]};
V={aux8(:)};
W={[aux22(:); b18(ones(160,1),:)]};
X={aux23(:)};
Y={[a12([ones(1,2400)],:)]};
Z={[a14([ones(1,2400)],:)]};
a={[a16([ones(1,2400)],:)]};
b={[a18([ones(1,2400)],:)]};
c={[a20([ones(1,2400)],:)]};
d={[a22([ones(1,2400)],:)]};
e={[a24([ones(1,2400)],:)]};
f={[a24([ones(1,2400)],:)]};
g={[a4([ones(1,2400)],:)]};
h={[a26([ones(1,2400)],:)]};
k={[a28([ones(1,2400)],:)]};

```

```

l={[a6([ones(1,2400)],:)];}
m={[a30([ones(1,2400)],:)];}
n={[a32([ones(1,2400)],:)];}
o={aux6(:)};
p={[a40([ones(1,2400)],:)];}
q={[a14([ones(1,2400)],:)];}
r={[a16([ones(1,2400)],:)];}
s={[a18([ones(1,2400)],:)];}
t={[a20([ones(1,2400)],:)];}
u={[a22([ones(1,2400)],:)];}
v={[a24([ones(1,2400)],:)];}
w={[a24([ones(1,2400)],:)];}
x={[a34([ones(1,2400)],:)];}
y={[a30([ones(1,2400)],:)];}
z={[a32([ones(1,2400)],:)];}

%=====
% In this step, the cells with values from the several
% simulations are assigned to structure I.
%
%
I=struct('num',A,'swtsh',B,'carry',C,'nlk',D,'chaff',E,'nois
e',F,'seltrac',G,'skal',H, ...

'ska2',J,'skv1',K,'skv2',L,'enra',M,'enrb',N,'tal',O,'ta2',P
,'tbl',Q,'tb2',R, ...

'tm',S,'tmv',T,'tv',U,'ttg',V,'sekhlim',W,'sekvlim',X,'ppc',
Y,'freqc',Z, ...

'antgainc',a,'HPc',b,'noibwc',c,'noifigc',d,'rrc',e,'numpulc
',f, ...

'nosangc',g,'envc',h,'nutfreqc',k,'serbwc',l,'crossc',m,'alp
hac',n, ...

'beam',o,'ppm',p,'freqm',q,'antgainm',r,'HPm',s,'noibwm',t,.
..
'noifigm',u,'rrm',v,'numpulm',w,'fslrm',x,'crossm',y,'alpham
',z);

%=====

% This section shows the user a GUI (Graphical User
% Interface)
% "Menu" to select a pre-programmed simulation from the
% initialization structure I.
% Appendix      has the simulations numbers associated with

```

```

% all model parameters.
% In this GUI ,missile and captive-carry simulations are
% runned, stored and ,if desired,
% graphed for a comparitive analysis.

menu

%=====
%
%Define po global variable. This variable plays an important
% role to set the exact
% moment of nulka rocket launching. Its parameters are :
% po= [ 'trigger time'(internal flag-allows change po value
% only once),
%       initial Nulka x-position,initial Nulka y-
%       position,initial Nulka z-position,
%       launching time].
% Initially all values are equal to zero.

po=[ 0 0 0 0 0];

%=====
%
% This section initializes the model parameters (Default)
%
% *** SEEKER
% azimuth
skal=I.skal(1,1);
ska2=I.ska2(1,1);
% elevation
skv1=I.skv1(1,1);
skv2=I.skv2(1,1);

% *** PROPORTIONAL NAVIGATION
% *** EFFECTIVE NAVIGATION RATIO
% azimuth
enra=I.enra(1,1);
% elevation
enrb=I.enrb(1,1);

% *** AUTOPILOT
% azimuth
tal=I.tal(1,1);
ta2=I.ta2(1,1);
% elevation
tb1=I.tb1(1,1);
tb2=I.tb2(1,1);

```

```

%     *** MISSILE DYNAMICS
% XY movement acceleration delay
%
tm=I.tm(1,1);

% Z movement acceleration delay
%
tmv=I.tmv(1,1);

% Velocity Change Delay
tv=I.tv(1,1);

%     *** Time-to-Go
ttg=I.ttg(1,1);

%     *** Seeker FOV(15 degrees for each side of LOS
%             in azimuth & elevation)
%
%
%
%     *** Values in radians
sekhlim=I.sekhlim(1,1);
sekvlim=I.sekvlim(1,1);

%Default parameters for running simulation

swtsh= I.swtsh(1,1);           %Simulation has a stopped target
                                %as default
carry=I.carry(1,1);           %Simulation has a missile as a
                                %threat as default
nlk=I.nlk(1,1);               %Simulation has no Nulka
                                %activated as default
chaff= I.chaff(1,1);           %Simulation has no chaff
                                %activated as default
noise= I.noise(1,1);           %Simulation has no noise in the
                                %seeker as default
num=I.num(1,1);                %Simulation number is set to 1 as
                                %default

=====
seltrac=I.seltrac(1,1);        %Simulation has a COSRO seeker as
                                %default
beam =I.beam(1,1);             %Simulation has a Gaussian beam
                                %shape as default

=====
```

```

% ***COSRO seeker initial parameters

ppc=I.ppc(1,1);
freqc=I.freqc(1,1);
antgainc=I.antgainc(1,1);
HPc=I.HPc(1,1);
noibwc=I.noibwc(1,1);
noifigc=I.noifigc(1,1);
rrc=I_rrc(1,1);
numpulc=I.numpulc(1,1);
nosangc=I.nosangc(1,1);
envc=I.envc(1,1);
nutfreqc=I.nutfreqc(1,1);
serbwc=I.serbwc(1,1);
crossc=I.crossc(1,1);
alphac=I.alphac(1,1);

% ***MONOPULSE seeker initial parameters

ppm=I.ppm(1,1);
freqm=I.freqm(1,1);
antgainm=I.antgainm(1,1);
HPm=I.HPm(1,1);
noibwm=I.noibwm(1,1);
noifigm=I.noifigm(1,1);
rrm=I_rrm(1,1);
numpulm=I.numpulm(1,1);
fslrm=I.fslrm(1,1);
crossm=I.crossc(1,1);
alpham=I.alphac(1,1);

%=====
%
% Define the seekers values/parameters of the vector
"current" that will be used
% for the noise generation in the Noise Generator Block.
%

db=[0];           %dB is just an auxiliary variable to set
                  %zeros in the "current" vector

current=[ [db([ones(1,22)])] I.ppc(1,1) I.freqc(1,1)
I.antgainc(1,1) I.HPc(1,1) ...
          I.noibwc(1,1) I.noifigc(1,1) I_rrc(1,1)
I.numpulc(1,1) I.nosangc(1,1)...
          I.envc(1,1) I.nutfreqc(1,1) I.serbwc(1,1)
I.crossc(1,1) I.alphac(1,1) ...
          I.ppm(1,1) I.freqm(1,1) I.antgainm(1,1)
I.HPm(1,1) I.noibwm(1,1) ...

```

```
I.noifigm(1,1) I.rrc(1,1) I.numpulm(1,1)
I.fslrm(1,1) ...
I.crossm(1,1) I.alpham(1,1)]';
```


APPENDIX C. FUNCTION CODE noise_error.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: noise_error.m
%%
%% Type: FUNCTION
%%
%% Purpose: This m-file calculates the noise that will be
%% added to the H.L.O.S(Horizontal Line of Sight)
%% and V.L.O.S (Vertical Line of Sight) in the Seeker
%% Dynamics Block. The theory behind this calculation is
%% explained on body of the thesis(Chap.II)
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
function y = noise_error(u)
%Create function noise_error
%
%=====
%
%                               Definitions

R = sqrt(u(26)^2+u(27)^2+u(28)^2+1e-10)*.3048;
% Range in meters (Units conversion)
% in order to work in the MKS system

if (u(29)==0),
    %If seeker is COSRO

aux = u(1:8);           % aux is an auxiliary variable
tek3 = u(9);            % Normalized offset angle
tc = u(10);             % Envelope correlation time (s)
fs = u(11);             % Nutation frequency (Hz)
beta = u(12);           % Servo bandwidth (Hz)
beam = u(30);           % Beam shape(0/Gaussian,1/sinc)

else
    %If seeker is Monopulse

```

```

aux = [u(15:22)]; % aux is an auxiliary variable
slr1 = u(23); % First side lobe Ratio (dB)

end;

Pt = aux(1)*1e3; % Peak Power (W)
freq = aux(2)*1e9; % Frequency (Hz)
G = 10^(aux(3)/10); % Antenna gain
HP = aux(4)*pi/180; % Half Power Beam Width-3dB (rd)
B = aux(5)*1e6; % Noise Bandwidth (Hz)
F = 10^(aux(6)/10); % Noise figure
dR = aux(7); % Range resolution (m)
n = aux(8); % Number of pulses integrated
RCS = u(13); % Target Radar Cross Section (m^2)
alfa = u(14); % One way atmospheric attenuation
% coeff(dB)

c = 3e8; % Speed of light
kT = 4.14e-21; % Product k(Boltzman's constant).T
% (300K)

%
%=====
%
%
% S/N Ratio for a single pulse
%
wl = c/freq; % Wavelength
N = kT*B*F; % Noise

aRdB = alfa*2*R/1000;
% Atmospheric Attenuation 2-W to R (dB)

aR = 10^(aRdB/10);
% Atmospheric Attenuation. 2-W to R

L = exp(-aR);
% Atmospheric loss 2-way

SNR = (Pt*G^2*RCS*wl^2*L)/((4*pi)^3*R^4*N); % S/N Ratio

if (u(29)==0), % If is is a COSRO seeker

% Poly Fit - Curves: Normalized offset angle X

Ks1/sqrt(Lk1) (and Ks1) (From Fig.14 on Chapter 2)
x1=[0.01,0.2,0.4,0.5,0.7,0.9,1];

% Cosro/Gaussian/Thermal

```

```

y1=[0.01,0.5,0.95,1,0.95,0.8,0.75];
p1=polyfit(x1,y1,5);

if (beam==0),      % If is a Gaussian beam shape for the
                  % COSRO Seeker
    ptet = p1;
    psci = [2.77,0.0001];

    % Cosro/Gaussian/Scintillation

else
    % If is a Sinc beam shape for the COSRO Seeker

    x2=[0.1,0.4,0.5,0.7,0.8,1];
    y2=[0.25,0.9,1,1.15,1.2,1.05];
    p2=polyfit(x2,y2,5);

    ptet =p2;

%Cosro/Sinc/Thermal

    x3=[0.1,0.4,0.6,0.7,0.8];
    y3=[0.25,1.2,1.9,2.5,3.3];
    p3=polyfit(x3,y3,5);

    psci =p3;

%Cosro/Sinc/Scintillation

end;

KsLk    = polyval(ptet,tek3);
Ks     = polyval(psci,tek3);

%                                         Standard Deviations

stet    = HP/(KsLk*sqrt(SNR*n));          % Teta
sci    = 0.225*HP*sqrt(beta/tc)/(fs*Ks);  % Scintilation
sd     = sqrt(stet^2+sci^2);                % General

else                               % Monopulse

% Poly Fit - Curve: First Sidelobe Ratio X Km (Gaussian)

x4=[15,20,24,30,40];
y4=[1.71,1.85,1.9,2,2.15];
p4=polyfit(x4,y4,5);

pm    = p4;

```

```

Km    = polyval(pm,slrl);

                                % Standard Deviation

sd    = HP/(Km*sqrt(2*SNR*n));

end;
%
%=====
%
%
% Errors (azimuth => dfi, elevation => dteta)
%
% Note: The errors have a Gaussian distribution. The mean
%       should be zero (<erro>=0), and the standard
%       deviation given by the expressions above,
%       ie error = Normal(0,sd). Therefore, we can
%       randomly generate an "error" from a
%       Normal distribution (0,sd) for each angle
%       measured. This "error", when summed to the
%       actual value (VLOS,HLOS)causes the seeker to
%       have a behavior statistically similar to the
%       real system.

ph = randn;
pv = randn;

y=zeros(2,1);

y(1,1) = sd*ph; % Horizontal Angular Error
y(2,1) = sd*pv; % Vertical Angular Error

```

APPENDIX D. FUNCTION CODE nulka.m

```
% Project: ASCM - Missile/Captive-Carry Simulation
%
% Name: nulka.m
%
% Type:FUNCTION
%
% Purpose: This m-file is used to calculate the Nulka
%           rocket position in a simulation.
%           The Nulka rocket launching time is a function
%           of the time_to_go. The function is
%           located in the Nulka Generator Block.
%           The Nulka intial position is on board of the
%           ship and its final position is 45 degrees above
%           the sea level at a height of 400 ft.
%
% Date Last Modified: 07/31/98
%
% By: LT Wagner A. de Lima Goncalves
%
function p=nulka(u)
%Creates the nulka function

global po
% Created in the initialization code initial.m.
% The matrix po play a important role in the Nulka
% lauching.Its structure is:
%     po= ['trigger time',Nulka x-position,Nulka y-position,
%           Nulka z-position,launching time']

=====
%
%           Nulka Position Calculation
%
```

% Initially the Nulka position is the same of the ship

```

Xn=u(1); %Nulka x-position = Ship x-position
Yn=u(2); %Nulka y-position = Ship y-position
Zn=u(3); %Nulka z-position = Ship z-position
Vs=u(4);
% Nulka Horizontal Velocity = Ship Velocity
% Assume values of 50/3 ft/sec (10 Knts) or
% 20ft/sec (12knts)

Vn=u(5);
%Nulka Ascension Magnitude Velocity (100 ft/sec)

flag=u(6);
% flag - allows the Nulka rocket be launched only once when
% time_to_go is reached.

t=u(7);
%Introduces the current simulation time

if flag==1 %Time_to_go reached ?

    if po(1)==0
    %Yes,Time_to_go reached.'Trigger time' reached ?

        po(2:5)=[Xn,Yn,Zn,t];
    % No,'Trigger time' not reached yet.Freeze the current
    % ship position and simulation time.

        po(1)=1;
    %Set the 'Trigger time' to one.'Trigger time' reached.

end

dt=t-po(5);
% Yes,'Trigger time reached',calculate the time difference
% between the current simulation time and the instant that
% 'Trigger time' was reached.

% Calculate the current Nulka position

Xn=po(2)+Vs*dt;
Yn=po(3)+Vn*cos(1.309)*dt;
Zn=po(4)+Vn*sin(1.309)*dt;

% Nulka altitude checking

```

```

ttop=400/(Vn*sin(1.309));
% Maximum time to reach the height of 400 ft with
% vertical component of the Nulka ascension magnitude
% velocity.

if dt>=ttop
% If the time difference is bigger than the maximum time
% to reach 400 ft height, keep the Nulka in a constant
% position in relation to the ship.

Xn=po(2)+Vs*ttop;
Yn=po(3)+Vn*cos(1.309)*ttop;
Zn=po(4)+Vn*sin(1.309)*ttop;
end

end
% Send the Nulka position back to the Nulka Generator
p(1)=Xn;
p(2)=Yn;
p(3)=Zn;

```


APPENDIX E. FUNCTION CODE menu.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: menu.m
%%
%% Type:FUNCTION
%%
%% Purpose: This m-file is a machine-generated
%% representation of a Handle Graphics object
%% and its children.
%% To reopen this object, just type the name of the M-file
%% at the MATLAB prompt.
%% The M-file and its associated MAT-file must be in your
%% path.
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
function fig = menu()

load menu

h0 = figure('Color',[1 1 1], ...
'Colormap',mat0, ...
'Name','Menu', ...
'PaperPosition',[0.25 2.5 8 6.000000000000001], ...
'PointerShapeCData',mat1, ...
'Position',[1 34 1024 687], ...
'Tag','Fig1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'ListboxTop',0, ...
'Position',mat2, ...
'Style','frame', ...
'Tag','Frame2');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...

```

```

'ListboxTop',0, ...
'Position',mat3, ...
'String','Beam shape', ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',mat4, ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',[380.4827586206897 11.17241379310345
230.896551724138 195.5172413793104], ...
[String',mat5, ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',[438.2068965517242 142.1379310344828
142.1379310344828 21.10344827586208], ...
[String','If noise =On and Tracker type=On', ...
'Style','text', ...
'Tag','StaticText5');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',mat6, ...
[String',mat7, ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontSize',14, ...
'ListboxTop',0, ...
'Position',mat8, ...
[String','ASCM Model Parameters', ...
'Style','text', ...
'Tag','StaticText6');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...

```

```

'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat9, ...
'String','Parameters ', ...
'Style','text', ...
'Tag','StaticText9');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat10, ...
'Position',mat11, ...
'String',mat12, ...
'Style','listbox', ...
'Tag','mono', ...
'Value',10);

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat13, ...
'Position',mat14, ...
'String',mat15, ...
'Style','listbox', ...
'Tag','ascm', ...
'Value',1);

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',[25.44827586206897 11.17241379310345
135.9310344827587 304.1379310344829], ...
'String','Beam shape', ...
'Style','frame', ...
'Tag','Frame1');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'HandleVisibility','off', ...
'ListboxTop',0, ...
'Position',[26.06896551724138 318.4137931034484
235.8620689655173 91.24137931034483], ...
'Style','frame', ...
'Tag','Frame1');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',[390.4137931034484 319.0344827586208
111.1034482758621 90.00000000000001], ...

```

```

'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',mat16, ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[447.5172413793105 219.1034482758621
37.24137931034483 19.86206896551725], ...
'String','Current Value', ...
'Style','text', ...
'Tag','StaticText11');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat17, ...
'ListboxTop',0, ...
'Position',mat18, ...
'String','0.25', ...
'Style','text', ...
'Tag','value1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[462.4137931034484 26.68965517241379
37.24137931034483 21.10344827586208], ...
'String','Current Value', ...
'Style','text', ...
'Tag','StaticText15');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat19, ...
'ListboxTop',0, ...
'Position',mat20, ...
'String','3000', ...
'Style','text', ...
'Tag','value3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...

```

```

'BackgroundColor',[1 1 1], ...
'Callback','update', ...
'ListboxTop',0, ...
'Position',mat21, ...
'String',mat22, ...
'Style','popupmenu', ...
'Tag','PopupMenul', ...
'Value',1);
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontSize',10, ...
'ListboxTop',0, ...
'Position',mat23, ...
'String','Select Missile Simulation', ...
'Style','text', ...
'Tag','StaticText1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','updatel,ascm', ...
'ListboxTop',0, ...
'Position',mat24, ...
'String','Run C.C Simulation', ...
'Tag','Pushbutton3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','store_simulation1', ...
'ListboxTop',0, ...
'Position',mat25, ...
'String','Store data', ...
'Tag','Pushbutton3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','ascm', ...
'ListboxTop',0, ...
'Position',[297.9310344827587 374.2758620689656
58.34482758620691 15.51724137931035], ...
'String','Run Simulation', ...
'Tag','Pushbutton1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','store_simulation', ...
'ListboxTop',0, ...
'Position',mat26, ...
'String','Store data', ...
'Tag','Pushbutton2');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','save gold,close', ...

```

```

'ListboxTop',0, ...
'Position',mat27, ...
'String','Exit', ...
'Tag','Pushbutton1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','simulations_plot', ...
'ListboxTop',0, ...
'Position',[528.2068965517243 374.896551724138
55.24137931034484 14.27586206896552], ...
'String','Plot ', ...
'Tag','Pushbutton4');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[471.1034482758622 173.7931034482759
90.0000000000001 18.62068965517242], ...
'String','Monopulse Parameters', ...
'Style','text', ...
'Tag','StaticText7');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[249.5172413793104 173.7931034482759
85.65517241379313 19.24137931034483], ...
'String','COSRO Parameters', ...
'Style','text', ...
'Tag','StaticText4');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',[223.448275862069 143.3793103448276
128.4827586206897 22.3448275862069], ...
'String','If noise =On and Tracker type=Off', ...
'Style','text', ...
'Tag','StaticText5');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat28, ...
'String','Beam shape', ...
'Style','text', ...

```

```

'Tag','StaticText10');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat29, ...
'String','Gaussian', ...
'Style','radiobutton', ...
'Tag','gaussian button');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat30, ...
'String','Sinc', ...
'Style','radiobutton', ...
'Tag','sinc button');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[168.8275862068966 88.75862068965519
50.27586206896553 15.51724137931035], ...
'String','Parameters ', ...
'Style','text', ...
'Tag','StaticText8');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat31, ...
'Position',mat32, ...
'String',mat33, ...
'Style','listbox', ...
'Tag','cosro', ...
'Value',14);
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[234.6206896551725 25.44827586206897
37.24137931034483 18.62068965517242], ...
'String','Current Value', ...
'Style','text', ...
'Tag','StaticText13');
h1 = uicontrol('Parent',h0, ...

```

```

'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback',mat34, ...
'ListboxTop',0, ...
'Position',mat35, ...
'String','0.055', ...
'Style','text', ...
'Tag','value2');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat36, ...
'String','Basic Settings', ...
'Style','text', ...
'Tag','StaticText2');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat37, ...
'String','Noise', ...
'Style','text', ...
'Tag','StaticText3');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat38, ...
'String','On', ...
'Style','radiobutton', ...
'Tag','noise button');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat39, ...
'String','On', ...
'Style','radiobutton', ...
'Tag','swtsh button', ...
'Value',1);

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...

```

```

'ListboxTop',0, ...
'Position',[29.79310344827587 253.8620689655173
44.6896551724138 15.51724137931035], ...
'String','Target', ...
'Style','text', ...
'Tag','StaticText3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat40, ...
'String','On', ...
'Style','radiobutton', ...
'Tag','carry button');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat41, ...
'String','Missile', ...
'Style','text', ...
'Tag','StaticText3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat42, ...
'String','On', ...
'Style','radiobutton', ...
'Tag','nlk button', ...
'Value',1);
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat43, ...
'String','Nulka', ...
'Style','text', ...
'Tag','StaticText3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat44, ...

```

```

'String','On', ...
'Style','radiobutton', ...
'Tag','chaff button');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat45, ...
'String','Chaff', ...
'Style','text', ...
'Tag','StaticText3');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[98.0689655172414 86.89655172413795
44.6896551724138 18.62068965517242], ...
'String','On', ...
'Style','radiobutton', ...
'Tag','seltrac button');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',[31.0344827586207 86.89655172413795
44.6896551724138 19.24137931034483], ...
'String','Tracker type', ...
'Style','text', ...
'Tag','StaticText3');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',mat46, ...
'String','Off=original target On=ship', ...
'Style','text', ...
'Tag','StaticText17');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',mat47, ...
'String','Off=Captive Carry On=missile', ...
'Style','text', ...
'Tag','StaticText18');

h1 = uicontrol('Parent',h0, ...

```

```

'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',mat48, ...
'String','Off=No nulka On=nulka on', ...
'Style','text', ...
'Tag','StaticText19');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',mat49, ...
'String','Off=chaff off On=chaff on', ...
'Style','text', ...
'Tag','StaticText20');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'ListboxTop',0, ...
'Position',mat50, ...
'String','Off=No seeker noise On=seeker noise', ...
'Style','text', ...
'Tag','StaticText21');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'HorizontalAlignment','left', ...
'ListboxTop',0, ...
'Position',mat51, ...
'String','Off=Cosro On=Monopulse', ...
'Style','text', ...
'Tag','StaticText22');
if nargout > 0, fig = h0; end

```


APPENDIX F. FUNCTION CODE simulations_plot.m

```

function fig = simulations_plot()

load simulations_plot

h0 = figure('Color',[0 0.501960784313725 1], ...
'Colormap',mat0, ...
'PaperOrientation','landscape', ...
'PaperPosition',[1.5 1.25 8 6.0000000000000001], ...
'PointerShapeCData',mat1, ...
'Position',[1 34 1024 687], ...
'Renderer','zbuffer', ...
'RendererMode','manual', ...
'Tag','Simulations Plot');

h1 = uicontrol('Parent',h0, ...
'Units','points', ...
>ListboxTop',0, ...
'Position',[361.8620689655173 243.9310344827587
216.6206896551725 128.4827586206897], ...
'Style','frame', ...

```

```

'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'ListboxTop',0, ...
'Position',[360.0000000000001 45.31034482758621
216.6206896551725 128.4827586206897], ...
'Style','frame', ...
'Tag','Frame1');
h1 = uicontrol('Parent',h0, ...
'ListboxTop',0, ...
'Style','text', ...
'Tag','ZOOMFigureState', ...
'UserData',mat2, ...
'Visible','off');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback','plot1', ...
'Position',[386.6896551724139 265.6551724137931
179.3793103448276 76.34482758620692], ...
[String',mat3, ...
'Style','listbox', ...
'Tag','Listbox1', ...
'Value',1];
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 1], ...
'Callback','plot2', ...
'Position',[382.344827586207 68.27586206896554
183.7241379310345 75.72413793103451], ...
[String',mat4, ...
'Style','listbox', ...
'Tag','Listbox2', ...
'Value',1];
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 0.501960784313725], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat5, ...
[String','Captive-Carry Simulation', ...
'Style','text', ...
'Tag','StaticText3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 0.501960784313725], ...
'FontWeight','bold', ...
'ListboxTop',0, ...
'Position',mat6, ...

```

```

'String','Missile Simulation', ...
'Style','text', ...
'Tag','StaticText2');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[1 1 0.501960784313725], ...
'FontSize',12, ...
'ListboxTop',0, ...
'Position',[411.5172413793104 388.5517241379311
121.0344827586207 18], ...
'String','Simulation Graphs ', ...
'Style','text', ...
'Tag','StaticText1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'BackgroundColor',[0 0.501960784313725 1], ...
'ListboxTop',0, ...
'Position',mat7, ...
'Style','frame', ...
'Tag','Frame3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','grid', ...
'ListboxTop',0, ...
'Position',[426.4137931034484 211.6551724137931
37.24137931034483 12.41379310344828], ...
'String','Grid', ...
'Tag','PushButton1');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','close', ...
'ListboxTop',0, ...
'Position',[426.4137931034484 198 37.24137931034483
12.41379310344828], ...
'String',mat8, ...
'Tag','PushButton2');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','zoom on', ...
'ListboxTop',0, ...
'Position',[492.2068965517242 211.6551724137931
37.24137931034483 12.41379310344828], ...
'String',mat9, ...
'Tag','PushButton3');
h1 = uicontrol('Parent',h0, ...
'Units','points', ...
'Callback','print -dwinc', ...
'ListboxTop',0, ...
'Position',mat10, ...

```

```

'String',mat11, ...
'Tag','PushButton4');
h1 = axes('Parent',h0, ...
'View',[40 40], ...
'CameraUpVector',[0 0 1], ...
'Color',[1 1 1], ...
'ColorOrder',mat12, ...
'Position',[0.13 0.5810982658959537 0.3270231213872832
0.3439017341040462], ...
'XColor',[0 0 0], ...
'XGrid','on', ...
'YColor',[0 0 0], ...
'YGrid','on', ...
'ZColor',[0 0 0], ...
'ZGrid','on');
h2 = line('Parent',h1, ...
'Color',[1 0 0], ...
'LineStyle','none', ...
'Marker','.', ...
'XData',mat13, ...
'YData',mat14, ...
'ZData',mat15);
h2 = line('Parent',h1, ...
'Color',[1 0 1], ...
'LineStyle','none', ...
'Marker','+', ...
'XData',mat16, ...
'YData',mat17, ...
'ZData',mat18);
h2 = line('Parent',h1, ...
'Color',[0 1 0], ...
'LineStyle','none', ...
'Marker','*', ...
'XData',mat19, ...
'YData',mat20, ...
'ZData',mat21);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','right', ...
'Position',[749441.8477667256 -7776.386765283653
1067.849833795002], ...
[String','x-axis', ...
'VerticalAlignment','top');
set(get(h2,'Parent'),'XLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...

```

```

'Position',[811193.4335722937 -7159.744849112582
1079.837853354824], ...
'String','y-axis', ...
'VerticalAlignment','top');
set(get(h2,'Parent'),'YLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','center', ...
'Position',[615184.7636742736 -7765.888135126959
1206.256968712944], ...
'Rotation',90, ...
'String','z-axis', ...
'VerticalAlignment','baseline');
set(get(h2,'Parent'),'ZLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','center', ...
'Position',[659080.864378504 -6215.741066851441
1350.113203430804], ...
'String','Scenario 3-D OverView', ...
'VerticalAlignment','bottom');
set(get(h2,'Parent'),'Title',h2);
h1 = axes('Parent',h0, ...
'View',[40 40], ...
'CameraUpVector',[0 0 1], ...
'Color',[1 1 1], ...
'ColorOrder',mat22, ...
'Position',[0.13 0.11 0.3270231213872832
0.3439017341040462], ...
'XColor',[0 0 0], ...
'XGrid','on', ...
'YColor',[0 0 0], ...
'YGrid','on', ...
'ZColor',[0 0 0], ...
'ZGrid','on');
h2 = line('Parent',h1, ...
'Color',[1 0 0], ...
'LineStyle','none', ...
'Marker','.', ...
'XData',mat23, ...
'YData',mat24, ...
'ZData',mat25);
h2 = line('Parent',h1, ...
'Color',[1 0 1], ...
'LineStyle','none', ...
'Marker','+', ...
'XData',mat26, ...

```

```

'YData',mat27, ...
'ZData',mat28);
h2 = line('Parent',h1, ...
'Color',[0 1 0], ...
'LineStyle','none', ...
'Marker','*', ...
'XData',mat29, ...
'YData',mat30, ...
'ZData',mat31);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','right', ...
'Position',[749441.8477667255 -31105.54706113461
5339.249168975011], ...
[String,'x-axis', ...
'VerticalAlignment','top');
set(get(h2,'Parent'),'XLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'Position',[811193.4335722935 -28638.97939645033
5399.18926677412], ...
[String,'y-axis', ...
'VerticalAlignment','top');
set(get(h2,'Parent'),'YLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','center', ...
'Position',[610809.5151661402 -31210.40331688718
6031.284843564718], ...
'Rotation',90, ...
[String,'z-axis', ...
'VerticalAlignment','baseline');
set(get(h2,'Parent'),'ZLabel',h2);
h2 = text('Parent',h1, ...
'Color',[0 0 0], ...
'HandleVisibility','off', ...
'HorizontalAlignment','center', ...
'Position',[659080.864378504 -24862.96426740577
6750.56601715402], ...
[String,'Scenario 3-D OverView', ...
'VerticalAlignment','bottom');
set(get(h2,'Parent'),'Title',h2);
if nargout > 0, fig = h0; end

```

APPENDIX G. FUNCTION CODE update.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%% Name: update.m
%% Type:FUNCTION
%% Purpose: This m-file updates the initialization
%% parameters of the ASCM Digital Model
%% for the missile simulation. It is located in the
%% Graphical User Interface GUI) "Menu" and it is activated
%% when the list (popup menu) containing the odd simulation
%% numbers is pressed on.
%% Date Last Modified: 07/31/98
%% By: LT Wagner A. de Lima Goncalves
nummm = get(findobj(gcf, 'Tag', 'PopupMenu1'), 'Value');
% Get the index of the popupmenu and
% assign it to the variable nummm

num=nummm;
%Assign the nummm value to num

db=[0];
%Create a single value matix to be
%used in the vector "current"

index=2*num-1;
%Create an odd index that will be used
%as a indexing parameter in the

```

```

%structure "store" to store the missile
%simulations.

po(1)=0;
%Re-initialization of the trigger time
%for the Nulka launching

%=====
%
%      Update the status radio-buttons in the GUI "Menu"
%
set(findobj(gcf,'Tag','swtsh button'), 'Value', I.swtsh(num));
% Update swtsh radio-button
set(findobj(gcf,'Tag','carry button'), 'Value', I.carry(num));
% Update carry radio-button
set(findobj(gcf,'Tag','nlk button'), 'Value', I.nlk(num));
% Update nlk radio-button
set(findobj(gcf,'Tag','chaff button'), 'Value', I.chaff(num));
% Update chaff radio-button
set(findobj(gcf,'Tag','noise button'), 'Value', I.noise(num));
% Update noise radio-button

set(findobj(gcf,'Tag','seltrac
button'), 'Value', I.seltrac(num));
% Update swtsh radio-button

%In this section, the radio-buttons located in the COSRO
parameters area in the GUI "Menu"
%"Menu".The logic is :
%Case 1:If there is noise in the seeker,the seeker is a
COSRO seeker and the beam shape has
%a Gaussian shape ==> set radio-button "on";
%Case 2:If there is noise in the seeker,the seeker is a
COSRO seeker and the beam shape has
%a Sinc function shape ==> set radio-button "on";
%Case 3:If there is noise in the seeker and the seeker is a
Monopulse seeker
%=> set radio-buttons "Gaussian" and "Sinc" "off".Also in
the case ,the indication of
%"Tracker Type" will be indicating "Monopulse" on.
%For I.noise(num)==0, the status radio-button will show
noise indication as "Off".(No noise
%in the seeker).

if I.noise(num)==1 & I.seltrac(num)==0 & I.beam(num)==0
% Case 1

```

```

        set(findobj(gcf,'Tag','gaussian
button'), 'Value', I.beam(num)+1);

        elseif I.noise(num)==1 & I.seltrac(num)==0 &
I.beam(num)==1    % Case 2
            set(findobj(gcf,'Tag','sinc
button'), 'Value', I.beam(num));
            set(findobj(gcf,'Tag','gaussian
button'), 'Value', I.beam(num)-1);
        else
% Case 3
            set(findobj(gcf,'Tag','sinc
button'), 'Value', I.beam(num));
            set(findobj(gcf,'Tag','gaussian
button'), 'Value', I.beam(num));
        end
%
%=====
%
%          Update Model Parameters
%
swtsh=I.swtsh(num);
carry=I.carry(num);
nlk=I.nlk(num);
chaff=I.chaff(num);
noise=I.noise(num);
seltrac=I.seltrac(num);
beam=I.beam(num);
ska1 = I.ska1(num);
ska2=I.ska2(num);
skv1=I.skv1(num);
skv2=I.skv2(num);
enra=I.enra(num);
enrb=I.enrb(num);
ta1=I.ta1(num);
ta2=I.ta2(num);
tb1=I.tb1(num);
tb2=I.tb2(num);
tm=I.tm(num);
tmv=I.tmv(num);
tv=I.tv(num);
ttg=I.ttg(num);
sekhlim=I.sekhlim(num);
sekvlim=I.sekvlim(num);
%
%=====
%
%          Update COSRO Seeker Parameters
%

```

```

ppc=I.ppc(num);
freqc=I.freqc(num);
antgainc=I.antgainc(num);
HPc=I.HPc(num);
noibwc=I.noibwc(num);
noifigc=I.noifigc(num);
rrc=I_rrc(num);
numpulc=I.numpulc(num);
nosangc=I.nosangc(num);
envc=I.envc(num);
nutfreqc=I.nutfreqc(num);
serbwc=I.serbwic(num);
crossc=I.crossc(num);
alphac=I.alphac(num);
%
%=====
%
%          Update Monopulse Seeker Parameters
%
ppm=I.ppm(num);
freqm=I.freqm(num);
antgainm=I.antgainm(num);
HPm=I.HPm(num);
noibwm=I.noibwm(num);
noifigm=I.noifigm(num);
rrm=I_rrm(num);
numpulm=I.numpulm(num);
fslrm=I.fslrm(num);
crossm=I.crossc(num);
alpham=I.alphac(num);
%
%=====
%
% Update the vector "current" that will be used for the
% noise generation in the
% Noise Generator Block.
%
current=[ [db([ones(1,22)])] I.ppc(num) I.freqc(num)
I.antgainc(num) I.HPc(num)...
I.noibwc(num) I.noifigc(num) I_rrc(num) I.numpulc(num)
I.nosangc(num)...
I.envc(num) I.nutfreqc(num) I.serbwic(num) I.crossc(num)
I.alphac(num)...
I.ppm(num) I.freqm(num) I.antgainm(num) I.HPm(num)
I.noibwm(num) ...
I.noifigm(num) I_rrc(num) I.numpulm(num) I.fslrm(num)
I.crossm(num)...
I.alpham(num) ];

```

APPENDIX H. FUNCTION CODE update1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%% Project: ASCM - Missile/Captive-Carry Simulation
%%% Name: update1.m
%%% Type:FUNCTION
%%% Purpose: This m-file updates the initialization
%%% parameters of the ASCM Digital Model for the
%%% Captive-Carry simulation. It is located in the
%%% Graphical User Interface (GUI) "Menu" and it is activated
%%% when the "Run C.C Simulation" button is pressed.
%%%
%%% Date Last Modified: 07/31/98
%%% By: LT Wagner A. de Lima Goncalves
%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% po(1)=0;
% Re-initilization of the trigger-time for the Nulka
% launching

index=index+1;
%Increment the indexing parameter of the "store" structure
%to store the even simulations(Captive-Carry).

I.carry(num)=0;
%Set the platform to be the captive-carry platform.

set(findobj(gcf,'Tag','carry button'),'Value',I.carry(num));
% Update radio-button status
% in the GUI "Menu"

%=====
%
```

Update Model Parameters

```

swtsh=I.swtsh(num);
carry=I.carry(num);
nlk=I.nlk(num);
chaff=I.chaff(num);
seltrac=I.seltrac(num);
beam=I.beam(num);
skal = I.skal(num);
ska2=I.ska2(num);
skv1=I.skv1(num);
skv2=I.skv2(num);
enra=I.enra(num);
enrb=I.enrb(num);
ta1=I.ta1(num);
ta2=I.ta2(num);
tb1=I.tb1(num);
tb2=I.tb2(num);
tm=I.tm(num);
tmv=I.tmv(num);
tv=I.tv(num);
ttg=I.ttg(num);
sekhlim=I.sekhlim(num);
sekvlim=I.sekvlim(num);
%
%=====
%
%           Update COSRO Seeker Parameters
%
ppc=I.ppc(num);
freqc=I.freqc(num);
antgainc=I.antgainc(num);
HPc=I.HPc(num);
noibwc=I.noibwc(num);
noifigc=I.noifigc(num);
rrc=I_rrc(num);
numpulc=I.numpulc(num);
nosangc=I.nosangc(num);
envc=I.envc(num);
nutfreqc=I.nutfreqc(num);
serbwc=I.serbwc(num);
crossc=I.crossc(num);
alphac=I.alphac(num);
%
%=====
%
%           Update Monopulse Seeker Parameters
%
ppm=I.ppm(num);
freqm=I.freqm(num);
antgainm=I.antgainm(num);

```

```

HPm=I.HPm(num);
noibwm=I.noibwm(num);
noifigm=I.noifigm(num);
rrm=I.rrm(num);
numpulm=I.numpulm(num);
fslrm=I.fslrm(num);
crossm=I.crossc(num);
alpham=I.alphac(num);
%
%=====
% Update the vector "current" that will be used for the
% noise generation in the Noise Generator Block.
%
%
current=[ [db([ones(1,22)])] I.ppc(num) I.freqc(num)
I.antgainc(num) I.HPc(num)...
I.noibwc(num) I.noifigc(num) I.rrc(num) I.numpulc(num)
I.nosangc(num)...
I.envc(num) I.nutfreqc(num) I.serbwbc(num) I.crossc(num)
I.alphac(num)...
I.ppm(num) I.freqm(num) I.antgainm(num) I.HPm(num)
I.noibwm(num) ...
I.noifigm(num) I.rrc(num) I.numpulm(num) I.fslrm(num)
I.crossm(num) I.alpham(num) ]';

```


APPENDIX I. FUNCTION CODE store simulation.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: store_simulation.m
%%
%% Type:FUNCTION
%%
%% Purpose: This m-file stores the missile results generated
%% by the ASCM Digital Model in a Matlab structure
%% in order to construct graphs (plot1.m and plot2.)
%% which will allow simulation analysis as well as further
%% data manipulations for training, testing and
%% evaluation of Neural Networks. It is activated by
%% pressing the button "Store Data" on the GUI "Menu" in the
%% Missile frame.
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
mcell=cell(1,1);
miss_distance=min(min(R));
% Miss_distance calculation
mcell=miss_distance;

tcell=cell(1,1);
% Time storing for graphs
tcell=shpos(:,1);

%
% Define cells to be used in the data structure store
%
%                               Seeker Performance Data
%
scs1=cell(1,1);
scs2=cell(1,1);
scs3=cell(1,1);
scs4=cell(1,1);
scs5=cell(1,1);

```

```

scs6=cell(1,1);
%
%                               ASCM Digital Model Parameters
%
acs1=cell(1,1);
acs2=cell(1,1);
acs3=cell(1,1);
acs4=cell(1,1);
acs5=cell(1,1);
acs6=cell(1,1);
acs7=cell(1,1);
acs8=cell(1,1);
acs9=cell(1,1);
acs10=cell(1,1);
acs11=cell(1,1);
acs12=cell(1,1);
acs13=cell(1,1);
acs14=cell(1,1);
acs15=cell(1,1);
acs16=cell(1,1);
%
%                               COSRO Seeker Parameters
%
ccs1=cell(1,1);
ccs2=cell(1,1);
ccs3=cell(1,1);
ccs4=cell(1,1);
ccs5=cell(1,1);
ccs6=cell(1,1);
ccs7=cell(1,1);
ccs8=cell(1,1);
ccs9=cell(1,1);
ccs10=cell(1,1);
ccs11=cell(1,1);
ccs12=cell(1,1);
ccs13=cell(1,1);
ccs14=cell(1,1);
ccs15=cell(1,1);
%
%                               Monopulse Seeker Parameters
%
mcs1=cell(1,1);
mcs2=cell(1,1);
mcs3=cell(1,1);
mcs4=cell(1,1);
mcs5=cell(1,1);
mcs6=cell(1,1);
mcs7=cell(1,1);
mcs8=cell(1,1);

```

```

mcs9=cell(1,1);
mcs10=cell(1,1);
mcs11=cell(1,1);
%
%                               Threat Position
%
thcs1=cell(1,1);
thcs2=cell(1,1);
thcs1=cell(1,1);
%
%                               Target Position
%
tcs1=cell(1,1);
tcs2=cell(1,1);
tcs3=cell(1,1);
%
%                               Nulka Position
%
ncs1=cell(1,1);
ncs2=cell(1,1);
ncs3=cell(1,1);
%
%=====
%
%Assign values to the cells used in the data structure store
%
%                               Seeker Performance Data
%
scs1=shpos (:,2);
scs2=svpos (:,2);
scs3=shrate (:,2);
scs4=svrate (:,2);
scs5=shacc (:,2);
scs6=svacc (:,2);
%
%                               ASCM Digital Model Parameters
%
acs1=I.skal(num);
acs2=I.ska2(num);
acs3=I.skv1(num);
acs4=I.skv2(num);
acs5=I.enra(num);
acs6=I.enrb(num);
acs7=I.ta1(num);
acs8=I.ta2(num);
acs9=I.tb1(num);
acs10=I.tb2(num);
acs11=I.tm(num);
acs12=I.tmv(num);

```

```

acs13=I.tv(num);
acs14=I.ttg(num);
acs15=I.sekhlim(num);
acs16=I.sekvlim(num);
%
%
%                                COSRO Seeker Parameters
%
ccs1=I.ppc(num);
ccs2=I.freqc(num);
ccs3=I.antgainc(num);
ccs4=I.HPc(num);
ccs5=I.noibwc(num);
ccs6=I.noifigc(num);
ccs7=I.rrc(num);
ccs8=I.numpulc(num);
ccs9=I.nosangc(num);
ccs10=I.envc(num);
ccs11=I.nutfreqc(num);
ccs12=I.serbw(c(num);
ccs13=I.crossc(num);
ccs14=I.alphac(num);
ccs15=I.beam(num);
%
%
%                                Monopulse Seeker Parameters
%
mcs1=I.ppm(num);
mcs2=I.freqm(num);
mcs3=I.antgainm(num);
mcs4=I.HPm(num);
mcs5=I.noibwm(num);
mcs6=I.noifigm(num);
mcs7=I.rrm(num);
mcs8=I.numpulm(num);
mcs9=I.fslrm(num);
mcs10=I.crossc(num);
mcs11=I.alphac(num);
%
%
%                                Threat Position
%
thcs1=m xpos(:,2);
thcs2=m ypos(:,2);
thcs3=m zpos(:,2);
%
%
%                                Target Position
%
tcs1=t xpos(:,2);
tcs2=t ypos(:,2);
tcs3=t zpos(:,2);
%

```

```

%
Nulka Position
%
ncs1=nxpos (:,2);
ncs2=nypos (:,2);
ncs3=nzpos (:,2);
%
%=====
%Structure store

store(index)=struct('miss',mcell,'time',tcell,'shpos',scs1,
'svpos',scs2,'shrate',scs3, ...
'svrate',scs4,'shacc', scs5,'svacc',scs6,'skal',acs1, ...
'ska2',acs2,'skv1',acs3,'skv2',acs4,'enra',acs5,'enrb',acs6,
...
'ta1',acs7,'ta2',acs8,'tb1',acs9,'tb2',acs10,'tm',acs11, ...
'tmv',acs12,'tv',acs13,'ttg',acs14,'sekhlim',acs15, ...
'sekvlim',acs16,'ppc',ccs1,'freqc',ccs2,'antgainc',ccs3, ...
'HPC',ccs4,'noibwc',ccs5,'noifigc',ccs6,'rrc',ccs7,'numpulc',
'ccs8, ...
'nosangc',ccs9,'envc',ccs10,'nutfreqc',ccs11, ...
'serbwc',ccs12,'crossc',ccs13,'alphac',ccs14,'beam',ccs15, ..
'ppm',mcs1,'freqm',mcs2,'antgainm',mcs3,'HPm',mcs4, ...
'noibwm',mcs5,'noifigm',mcs6,'rrm',mcs7,'numpulm',mcs8,'fslm
',mcs9, ...
'crossm',mcs10,'alpham',mcs11,'mxpos',thcs1,'mypos',thcs2, ...
'mzpos',thcs3,'txpos',tcs1,'typos',tcs2,'tzpos',tcs3, ...
'nxpos',ncs1,'nypos',ncs2,'nzpos',ncs3);

```


APPENDIX J. FUNCTION CODE store_simulation1.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: store_simulation1.m
%%
%% Type:FUNCTION
%%
%% Purpose: This m-file stores the captive-carry results
%% generated by the ASCM Digital Model in a Matlab structure
%% in order to construct graphs
%% (plot1.m and plot2.) which will allow simulation
%% analysis as well as further data manipulations
%% for training, testing and evaluation of a Neural Network.
%% It is activated by pressing the button "Store Data"
%% on the GUI "Menu" in the Captive-carry frame.
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
mcell2=cell(1,1);
miss_distance2=min(min(R));
% Miss_distance calculation
mcell2=miss_distance2;

tcell2=cell(1,1);
% Time storing for graphs
tcell2=shpos(:,1);

%
Define cells to be used in the data structure store

%
Seeker Performance Data

scs12=cell(1,1);
scs22=cell(1,1);
scs32=cell(1,1);
scs42=cell(1,1);
scs52=cell(1,1);

```

```
scs62=cell(1,1);
%
%
%                               ASCM Digital Model Parameters
%
acs12=cell(1,1);
acs22=cell(1,1);
acs32=cell(1,1);
acs42=cell(1,1);
acs52=cell(1,1);
acs62=cell(1,1);
acs72=cell(1,1);
acs82=cell(1,1);
acs92=cell(1,1);
acs102=cell(1,1);
acs112=cell(1,1);
acs122=cell(1,1);
acs132=cell(1,1);
acs142=cell(1,1);
acs152=cell(1,1);
acs162=cell(1,1);
%
%
%                               COSRO Seeker Parameters
%
ccs12=cell(1,1);
ccs22=cell(1,1);
ccs32=cell(1,1);
ccs42=cell(1,1);
ccs52=cell(1,1);
ccs62=cell(1,1);
ccs72=cell(1,1);
ccs82=cell(1,1);
ccs92=cell(1,1);
ccs102=cell(1,1);
ccs112=cell(1,1);
ccs122=cell(1,1);
ccs132=cell(1,1);
ccs142=cell(1,1);
ccs152=cell(1,1);
%
%
%                               Monopulse Seeker Parameters
%
mcs12=cell(1,1);
mcs22=cell(1,1);
mcs32=cell(1,1);
mcs42=cell(1,1);
mcs52=cell(1,1);
mcs62=cell(1,1);
mcs72=cell(1,1);
mcs82=cell(1,1);
```

```

mcs92=cell(1,1);
mcs102=cell(1,1);
mcs112=cell(1,1);
%
%                               Threat Position
%
thcs12=cell(1,1);
thcs22=cell(1,1);
thcs32=cell(1,1);
%
%                               Target Position
%
tcs12=cell(1,1);
tcs22=cell(1,1);
tcs32=cell(1,1);
%
%                               Nulka Position
%
ncs12=cell(1,1);
ncs22=cell(1,1);
ncs32=cell(1,1);
%
=====
%
% Assign values to the cells used in the data structure
% store
%
%                               Seeker Performance Data
%
scs12=shpos(:,2);
scs22=svpos(:,2);
scs32=shrate(:,2);
scs42=svrate(:,2);
scs52=shacc(:,2);
scs62=svacc(:,2);
%
%                               ASCM Digital Model Parameters
%
acs12=I.ska1(num);
acs22=I.ska2(num);
acs32=I.skv1(num);
acs42=I.skv2(num);
acs52=I.enra(num);
acs62=I.enrb(num);
acs72=I.ta1(num);
acs82=I.ta2(num);
acs92=I.tb1(num);
acs102=I.tb2(num);
acs112=I.tm(num);

```

```

acs122=I.tmv(num);
acs132=I.tv(num);
acs142=I.ttg(num);
acs152=I.sekhlim(num);
acs162=I.sekvlim(num);
%
%
%                                COSRO Seeker Parameters
%
ccs12=I.ppc(num);
ccs22=I.freqc(num);
ccs32=I.antgainc(num);
ccs42=I.HPc(num);
ccs52=I.noibwc(num);
ccs62=I.noifigc(num);
ccs72=I.rrc(num);
ccs82=I.numpulc(num);
ccs92=I.nosangc(num);
ccs102=I.envc(num);
ccs112=I.nutfreqc(num);
ccs122=I.serbwc(num);
ccs132=I.crossc(num);
ccs142=I.alphac(num);
ccs152=I.beam(num);
%
%
%                                Monopulse Seeker Parameters
%
mcs12=I.ppm(num);
mcs22=I.freqm(num);
mcs32=I.antgainm(num);
mcs42=I.HPm(num);
mcs52=I.noibwm(num);
mcs62=I.noifigm(num);
mcs72=I.rrm(num);
mcs82=I.numpulm(num);
mcs92=I.fslrm(num);
mcs102=I.crossc(num);
mcs112=I.alphac(num);
%
%
%                                Threat Position
%
thcs12=mxpos(:,2);
thcs22=mypos(:,2);
thcs32=mzpos(:,2);
%
%
%                                Target Position
%
tcs12=txpos(:,2);
tcs22=typos(:,2);
tcs32=tzpos(:,2);

```

```

%
%                               Nulka Position
%
ncs12=nxpos (:,2);
ncs22=nypos (:,2);
ncs32=nzpos (:,2);
%
%=====
%Structure store

store(index)=struct('miss',mcell2,'time',tcell2,'shpos',scs1
2,'svpos',scs22, ...
'shrate',scs32,'svrate',scs42,'shacc', scs52,'svacc',scs62, ...
'ska1',acs12,'ska2',acs22,'skv1',acs32,'skv2',acs42, ...
'enra',acs52,'enrb',acs62,'ta1',acs72,'ta2',acs82,'tb1',acs9
2, ...
'tb2',acs102,'tm',acs112,'tmv',acs122,'tv',acs132,'ttg',acs1
42, ...
'sekhl1m',acs152,'sekvl1m',acs162,'ppc',ccs12,'freqc',ccs22,
'antgainc',ccs32,'HPc',ccs42,'noibwc',ccs52,'noifigc',ccs62,
...
'rrc',ccs72,'numpulc',ccs82,'nosangc',ccs92,'envc',ccs102, ...
'nutfreqc',ccs112,'serbw',ccs122,'crossc',ccs132, ...
'alphac',ccs142,'beam',ccs152,'ppm',mcs12,'freqm',mcs22, ...
'antgainm',mcs32,'HPm',mcs42,'noibwm',mcs52,'noifigm',mcs62,
...
'rrm',mcs72,'numpulm',mcs82,'fslrm',mcs92, ...
'crossm',mcs102,'alpham',mcs112,'mxpos',thcs12,'mypos',thcs2
2, ...
'mzpos',thcs32,'txpos',tcs12,'typos',tcs22,'tzpos',tcs32, ...
'n xpos',ncs12,'nypos',ncs22,'nzpos',ncs32);

%
I.carry(num)=1; % Set back the threat as a missile
po(1)=0;          % Re-initializtes the Nulka "trigger time"
index=index-1;   % Set back the storage index for the missile
% simulation

```


APPENDIX K. FUNCTION CODE plot1.m

```
% Project: ASCM - Missile/Captive-Carry Simulation
%
% Name: plot1.m
%
% Type:FUNCTION
%
% Purpose: This m-file uses the values stored by the
% function store_simulation.m for plotting the missile
% simulation. The graphs show the missile,target, and
% Nulka trajectory as well as the seeker
% performance(position, angular rate and acceleration)
% on the GUI "Simulations_plot".This function is
% activated when the list (popup menu) is selected on the
% GUI.
%
% Date Last Modified: 07/31/98
%
% By: LT Wagner A. de Lima Goncalves
%
% Assign the values stored in "store" structure at the index
% defined in the the GUI Menu(by update.m function) for the
% variables below to plot the missile simulation graphs.

paux1=store(index).mxpos;
% Store the missile x position;

paux2=store(index).mypos;
% Store the missile y position;

paux3=store(index).mzpos;
% Store the missile z position;

paux4=store(index).txpos;
% Store the target x position;

paux5=store(index).typos;
% Store the target y position;
```

```

paux6=store(index).tzpos;
% Store the target z position;

paux7=store(index).nxpos;
% Store the Nulka x position;

paux8=store(index).nypos;
% Store the Nulka y position;

paux9=store(index).nzpos;
% Store the Nulka z position;

paux10=store(index).shpos;
% Store the seeker horizontal position;

paux11=store(index).svpos;
% Store the seeker horizontal position;

paux12=store(index).shrate;
% Store the seeker horizontal rate;

paux13=store(index).svrate;
% Store the seeker vertical rate;

paux14=store(index).shacc;
% Store the seeker horizontal acceleration;

paux15=store(index).svacc;
% Store the seeker horizontal rate;

ptime=store(index).time;
% Store time of the simulation;

mdout=int2str(min(min(store(index).miss)));
% Calculate the miss-distance to be presented
% on the missile graphs.

nn=index;
%Assign the value of the current

%index (update.m) for the variable nn

V2=get(gco,'Value');
%Get the index of the graph list(popup menu)

if V2==1,

```

```

% First graph:3-D missile,target, and Nulka
% trajectories

subplot(2,2,1)
plot3(paux1,paux2,paux3,'r.',paux4,paux5,paux6,'m
+',paux7,paux8,paux9,'g*')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title( 'Scenario 3-D OverView')
view(40,40)

elseif V2==2,
% Second graph:2-D missile,target, and Nulka
% trajectories (X-Y)

subplot(2,2,1)
plot(paux1,paux2,'r.',paux4,paux5,'m +',paux7,paux8,'g*')
xlabel('x-axis')
ylabel('y-axis')
title(['Simulation Number:',int2str(nn),...
' Threat - Target Plot X-Y with Miss
Distance=',mdout,'ft'])

elseif V2==3,
% Third graph:2-D missile,target, and Nulka
% trajectories (X-Z)

subplot(2,2,1)
plot(paux1,paux3,'r.',paux4,paux6,'m +',paux7,paux9,'g*')
xlabel('x-axis')
ylabel('z-axis')
title('Threat - Target Plot X-Z')

elseif V2==4,
% Fourth graph:Seeker Horizontal Position

subplot(2,2,1)
paux10=(180*paux10)/pi;
plot(ptime,paux10)
xlabel('time(sec)')
ylabel('Horizontal Position(degrees)')
title('Seeker Horizontal Position')

elseif V2==5,
% Fifth graph:Seeker Vertical Position
subplot(2,2,1)
paux11=(180*paux11)/pi;
plot(ptime,paux11)

```

```

xlabel('time(sec)')
ylabel('Vertical Position(degrees)')
title('Seeker Vertical Position')

elseif V2==6,
% Sixth graph:Seeker Horizontal Rate
subplot(2,2,1)
plot(ptime,paux12)
xlabel('time(sec)')
ylabel('Horizontal Rate(rad/s)')
title('Seeker Horizontal Rate')

elseif V2==7,
% Seventh graph:Seeker Vertical Rate
subplot(2,2,1)
plot(ptime,paux13)
xlabel('time(sec)')
ylabel('Vertical Rate(rad/s)')
title('Seeker Vertical Rate')

elseif V2==8,
% Eighth graph:Seeker Horizontal Acceleration
subplot(2,2,1)
plot(ptime,paux14)
xlabel('time(sec)')
ylabel('Horizontal Acceleration(rad/s^2)')
title('Seeker Horizontal Acceleration')

elseif V2==9,
% Nineth graph:Seeker Vertical Acceleration
subplot(2,2,1)
plot(ptime,paux15)
xlabel('time(sec)')
ylabel('Vertical Acceleration(rad/s^2)')
title('Seeker Vertical Acceleration')
else,
end

```

APPENDIX L. FUNCTION CODE plot2.m

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: plot2.m
%%
%% Type:FUNCTION
%%
%% Purpose: This m-file uses the values stored by the
%% function store_simulation1.m for plotting the
%% captive-carry simulation. The graphs show the captive-
%% carry, target, and Nulka trajectory as well as
%% the seeker performance(position,angular rate and
%% acceleration) on the GUI "Simulations_plot".This function
%% is activated when the list (popup menu) is selected
%% on the GUI.
%%
%% Date Last Modified: 07/31/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
%% Assign the values stored in "store" structure in the index
%% defined in the the GUI Menu (by update.m function)
%% for the variables below to plot the missile simulation
%% graphs.
%%
paux12=store(index+1).mxpos;
% Store the captive-carry x position;

paux22=store(index+1).mypos;
% Store the captive-carry y position;

paux32=store(index+1).mzpos;
% Store the captive-carry z position;

paux42=store(index+1).txpos;
% Store the target x position;

paux52=store(index+1).typos;
% Store the target y position;

```

```

paux62=store(index+1).tzpos;
% Store the target z position;

paux72=store(index+1).nxpos;
% Store the Nulka x position;

paux82=store(index+1).nypos;
% Store the Nulka y position;

paux92=store(index+1).nzpos;
% Store the Nulka z position;

paux102=store(index+1).shpos;
% Store the seeker horizontal position;

paux112=store(index+1).svpos;
% Store the seeker horizontal position;

paux122=store(index+1).shrate;
% Store the seeker horizontal rate;

paux132=store(index+1).svrate;
% Store the seeker vertical rate;

paux142=store(index+1).shacc;
% Store the seeker horizontal acceleration;

paux152=store(index+1).svacc;
% Store the seeker vertical acceleration;

ptime2=store(index+1).time;
% Store time of the simulation;

mdout2=int2str(min(min(store(index+1).miss)));
% Calculate the miss-distance to be presented
% on the captive-carry graphs.

nn2=index;
% Assign the value of the current
% index (update.m) for the variable nn

V2=get(gco,'Value');
% Get the index of the graph list(popup menu)

if V2==1,
% First graph:3-D captive-carry ,target, and Nulka
% trajectories

```

```

subplot(2,2,2)
plot3(paux12,paux22,paux32,'r.',paux42,paux52,paux62,'m
+',paux72,paux82,paux92,'g*')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
title( 'Scenario 3-D OverView')
view(40,40)

elseif V2==2,
% Second graph:2-D captive-carry ,target, and Nulka
% trajectories (X-Y)

subplot(2,2,2)
plot(paux12,paux22,'r.',paux42,paux52,'m
+',paux72,paux82,'g*')
xlabel('x-axis')
ylabel('y-axis')
title(['Simulation Number:',int2str(nn+1),...
        ' Threat - Target Plot X-Y with Miss
Distance=',mdout2,'ft'])

elseif V2==3,
% Third graph:2-D captive-carry ,target, and Nulka
% trajectories (X-Z)

subplot(2,2,2)
plot(paux12,paux32,'r.',paux42,paux62,'m
+',paux72,paux92,'g*')
xlabel('x-axis')
ylabel('z-axis')
title('Threat - Target Plot X-Z')

elseif V2==4,
% Fourth graph:Seeker Horizontal Position

subplot(2,2,2)
paux102=(180*paux102)/pi;
plot(ptime2,paux102)
xlabel('time(sec)')
ylabel('Horizontal Position(degrees)')
title('Seeker Horizontal Position')

elseif V2==5,
% Fifth graph:Seeker Vertical Position

subplot(2,2,2)
paux112=(180*paux112)/pi;

```

```

plot(ptime2,paux112)
xlabel('time(sec)')
ylabel('Vertical Position(degrees)')
title('Seeker Vertical Position')

elseif V2==6,
% Sixth graph:Seeker Horizontal Rate

subplot(2,2,2)
plot(ptime2,paux122)
xlabel('time(sec)')
ylabel('Horizontal Rate(rad/s)')
title('Seeker Horizontal Rate')

elseif V2==7,
% Seventh graph:Seeker Vertical Rate

subplot(2,2,2)
plot(ptime2,paux132)
xlabel('time(sec)')
ylabel('Vertical Rate(rad/s)')
title('Seeker Vertical Rate')

elseif V2==8,
% Eighth graph:Seeker Horizontal Acceleration
subplot(2,2,2)
plot(ptime2,paux142)
xlabel('time(sec)')
ylabel('Horizontal Acceleration(rad/s^2)')
title('Seeker Horizontal Acceleration')

elseif V2==9,
% Nineth graph:Seeker Vertical Acceleration

subplot(2,2,2)
plot(ptime2,paux152)
xlabel('time(sec)')
ylabel('Vertical Acceleration(rad/s^2)')
title('Seeker Vertical Acceleration')

else,
end

```

APPENDIX M. MISSILE SIMULATIONS I

APPENDIX N. MISSILE SIMULATIONS II

APPENDIX O. NEURAL NETWORK IMPLEMENTATION

```

%% Project: ASCM - Missile/Captive-Carry Simulation
%%
%% Name: Neural Network
%%
%% Type:SCRIPT
%%
%% Purpose: This compiles several codes used in Chapter
%%           VIII for neural network designing, training,
%%           testing and missile trajectory prediction from
%%           the captive-carry configuration.
%%
%% Date Last Modified: 09/11/98
%%
%% By: LT Wagner A. de Lima Goncalves
%%
%Training Set # 1

p=[store(1,1).range' store(1,3).range' store(1,5).range' store(
1,7).range' store(1,9).range'...
    store(1,11).range' store(1,13).range'
store(1,15).range' store(1,17).range' store(1,19).range'...
    store(1,21).range' store(1,23).range'
store(1,25).range' store(1,27).range' store(1,29).range';

store(1,1).shpos' store(1,3).shpos' store(1,5).shpos' store(1,7)
).shpos' store(1,9).shpos'...

store(1,11).shpos' store(1,13).shpos' store(1,15).shpos' store(
1,17).shpos' store(1,19).shpos'...

store(1,21).shpos' store(1,23).shpos' store(1,25).shpos' store(
1,27).shpos' store(1,29).shpos';

```

```

store(1,1).svpos' store(1,3).svpos' store(1,5).svpos' store(1,7)
).svpos' store(1,9).svpos'...

store(1,11).svpos' store(1,13).svpos' store(1,15).svpos' store(
1,17).svpos' store(1,19).svpos'...

store(1,21).svpos' store(1,23).svpos' store(1,25).svpos' store(
1,27).svpos' store(1,29).svpos'];

t=[store(1,1).mxpos' store(1,3).mxpos' store(1,5).mxpos' store(
1,7).mxpos' store(1,9).mxpos'...

store(1,11).mxpos' store(1,13).mxpos' store(1,15).mxpos' store(
1,17).mxpos' store(1,19).mxpos'...

store(1,21).mxpos' store(1,23).mxpos' store(1,25).mxpos' store(
1,27).mxpos' store(1,29).mxpos';
    store(1,1).mypos'
store(1,3).mypos' store(1,5).mypos' store(1,7).mypos' store(1,9)
).mypos'...

store(1,11).mypos' store(1,13).mypos' store(1,15).mypos' store(
1,17).mypos' store(1,19).mypos'...

store(1,21).mypos' store(1,23).mypos' store(1,25).mypos' store(
1,27).mypos' store(1,29).mypos';

store(1,1).mzpos' store(1,3).mzpos' store(1,5).mzpos' store(1,7)
).mzpos' store(1,9).mzpos'...

store(1,11).mzpos' store(1,13).mzpos' store(1,15).mzpos' store(
1,17).mzpos' store(1,19).mzpos'...

store(1,21).mzpos' store(1,23).mzpos' store(1,25).mzpos' store(
1,27).mzpos' store(1,29).mzpos'];

% Neural Network Creation

factorp=max(max(p));
p(1,:)=p(1,:)/factorp;

factort=max(max(t));
t=t/factort;

```

```

net=newff([minmax(p)],[10 3],{'logsig'
'purelin'},'Trainlm');
net.initParam='rands';
init(net);
net.performFcn='mse';
net.trainParam.goal=1e-6;
net.trainParam.show=10;
net.trainParam.epochs=500;

%Neural Network Training
%
net=train(net,p,t);

% Training Set #2
K=30;
p=[store(1,1+K).range' store(1,3+K).range' store(1,5+K).range'
store(1,7+K).range' store(1,9+K).range'...
    store(1,11+K).range' store(1,13+K).range'
store(1,15+K).range' store(1,17+K).range'
store(1,19+K).range'...
    store(1,21+K).range' store(1,23+K).range'
store(1,25+K).range' store(1,27+K).range'
store(1,29+K).range';

store(1,1+K).shpos' store(1,3+K).shpos' store(1,5+K).shpos' sto
re(1,7+K).shpos' store(1,9+K).shpos'...
    store(1,11+K).shpos' store(1,13+K).shpos' store(1,15+K).shpos'
store(1,17+K).shpos' store(1,19+K).shpos'...
    store(1,21+K).shpos' store(1,23+K).shpos' store(1,25+K).shpos'
store(1,27+K).shpos' store(1,29+K).shpos';

store(1,1+K).svpos' store(1,3+K).svpos' store(1,5+K).svpos' sto
re(1,7+K).svpos' store(1,9+K).svpos'...
    store(1,11+K).svpos' store(1,13+K).svpos' store(1,15+K).svpos'
store(1,17+K).svpos' store(1,19+K).svpos'...
    store(1,21+K).svpos' store(1,23+K).svpos' store(1,25+K).svpos'
store(1,27+K).svpos' store(1,29+K).svpos'];

t=[store(1,1+K).mxpos' store(1,3+K).mxpos' store(1,5+K).mxpos'
store(1,7+K).mxpos' store(1,9+K).mxpos'...

```

```

store(1,11+K).mxpos' store(1,13+K).mxpos' store(1,15+K).mxpos'
store(1,17+K).mxpos' store(1,19+K).mxpos' ...

store(1,21+K).mxpos' store(1,23+K).mxpos' store(1,25+K).mxpos'
store(1,27+K).mxpos' store(1,29+K).mxpos';
    store(1,1+K).mypos'
store(1,3+K).mypos' store(1,5+K).mypos' store(1,7+K).mypos' sto
re(1,9+K).mypos' ...

store(1,11+K).mypos' store(1,13+K).mypos' store(1,15+K).mypos'
store(1,17+K).mypos' store(1,19+K).mypos' ...

store(1,21+K).mypos' store(1,23+K).mypos' store(1,25+K).mypos'
store(1,27+K).mypos' store(1,29+K).mypos';

store(1,1+K).mzpos' store(1,3+K).mzpos' store(1,5+K).mzpos' sto
re(1,7+K).mzpos' store(1,9+K).mzpos' ...

store(1,11+K).mzpos' store(1,13+K).mzpos' store(1,15+K).mzpos'
store(1,17+K).mzpos' store(1,19+K).mzpos' ...

store(1,21+K).mzpos' store(1,23+K).mzpos' store(1,25+K).mzpos'
store(1,27+K).mzpos' store(1,29+K).mzpos'];

% Neural Network Creation

factorp=max(max(p));
p(1,:)=p(1,:)/factorp;

factort=max(max(t));
t=t/factort;

%Neural Network Training
%
net=train(net,p,t);

% Training Set #3

K=40;

p=[store(1,1+K).range' store(1,3+K).range' store(1,5+K).range'
store(1,7+K).range' store(1,9+K).range' ...
    store(1,11+K).range' store(1,13+K).range'
store(1,15+K).range' store(1,17+K).range'
store(1,19+K).range'...

```

```

    store(1,21+K).range' store(1,23+K).range'
store(1,25+K).range'store(1,27+K).range'
store(1,29+K).range';

store(1,1+K).shpos'store(1,3+K).shpos'store(1,5+K).shpos'sto
re(1,7+K).shpos'store(1,9+K).shpos'...

store(1,11+K).shpos'store(1,13+K).shpos'store(1,15+K).shpos'
store(1,17+K).shpos'store(1,19+K).shpos'...

store(1,21+K).shpos'store(1,23+K).shpos'store(1,25+K).shpos'
store(1,27+K).shpos'store(1,29+K).shpos';

store(1,1+K).svpos'store(1,3+K).svpos'store(1,5+K).svpos'sto
re(1,7+K).svpos'store(1,9+K).svpos'...

store(1,11+K).svpos'store(1,13+K).svpos'store(1,15+K).svpos'
store(1,17+K).svpos'store(1,19+K).svpos'...

store(1,21+K).svpos'store(1,23+K).svpos'store(1,25+K).svpos'
store(1,27+K).svpos'store(1,29+K).svpos'];

t=[store(1,1+K).mxpos'store(1,3+K).mxpos'store(1,5+K).mxpos'
store(1,7+K).mxpos'store(1,9+K).mxpos'...

store(1,11+K).mxpos'store(1,13+K).mxpos'store(1,15+K).mxpos'
store(1,17+K).mxpos'store(1,19+K).mxpos'...

store(1,21+K).mxpos'store(1,23+K).mxpos'store(1,25+K).mxpos'
store(1,27+K).mxpos'store(1,29+K).mxpos';
    store(1,1+K).mypos'
store(1,3+K).mypos'store(1,5+K).mypos'store(1,7+K).mypos'sto
re(1,9+K).mypos'...

store(1,11+K).mypos'store(1,13+K).mypos'store(1,15+K).mypos'
store(1,17+K).mypos'store(1,19+K).mypos'...

store(1,21+K).mypos'store(1,23+K).mypos'store(1,25+K).mypos'
store(1,27+K).mypos'store(1,29+K).mypos';

store(1,1+K).mzpos'store(1,3+K).mzpos'store(1,5+K).mzpos'sto
re(1,7+K).mzpos'store(1,9+K).mzpos'...

store(1,11+K).mzpos'store(1,13+K).mzpos'store(1,15+K).mzpos'
store(1,17+K).mzpos'store(1,19+K).mzpos'...

```

```

store(1,21+K).mzpos' store(1,23+K).mzpos' store(1,25+K).mzpos'
store(1,27+K).mzpos' store(1,29+K).mzpos'];

% Neural Network Creation

factorp=max(max(p));
p(1,:)=p(1,:)/factorp;

factort=max(max(t));
t=t/factort;

%Neural Network Training
%
net=train(net,p,t);
%=====

% Neural Network Graphing (Missile Experiment)

j=input('Enter the j value. "j" ranges from 1 to 8 ---> ');
k= (j-1)*10;
ind=5+k;
p=[store(1,ind).range';
store(1,ind).shpos';
store(1,ind).svpos'];

factorp=max(max(p));
p(1,:)=p(1,:)/factorp;

a=sim(net,p);

figure(1)
subplot(3,1,1)
plot3(store(1,ind).mxpos,store(1,ind).mypos,store(1,ind).mzpos,'r-',...
      a(1,:)*factorp,a(2,:)*factorp,a(3,:)*factorp,'b-')
view(28,26)
orient tall
grid
xlabel('down range')
ylabel('cross range')
zlabel('altitude')
legend('missile','prediction')
title([' Missile Trajectory Comparison - Actual x NN
Prediction Simulation 'num2str(ind)])

```

```

subplot(3,2,3)
plot(store(1,ind).mxpos,store(1,ind).mypos,'r-.',...
     a(1,:)*factorp,a(2,:)*factorp,'b-')
grid
xlabel('down range')
ylabel('cross range')
%legend('missile','prediction')

subplot(3,2,4)

plot(store(1,ind).mxpos,store(1,ind).mzpos,'r-.',...
     a(1,:)*factorp,a(3,:)*factorp,'b-')
grid
xlabel('down range')
ylabel('altitude')
%legend('missile','prediction')

subplot(3,1,3)

xx1=store(1,ind).mxpos';
xx2=a(1,:)*factorp;
yy1=store(1,ind).mypos';
yy2=a(2,:)*factorp;
zz1=store(1,ind).mzpos';
zz2=a(3,:)*factorp;

ms_error=sqrt((xx1-xx2).^2+(yy1-yy2).^2+(zz1-zz2).^2);

plot(store(1,ind).time,ms_error)
xlabel('simulation Time (sec)')
ylabel('Mean Square Error (ft)')
grid

figure(2)

subplot(3,1,1)
plot(store(1,ind).time,store(1,ind).shpos)
xlabel('Time(sec)')
ylabel('Seeker Hor. Position(deg)')
title([' Neural Network Inputs - Simulation ' num2str(ind)])
grid

subplot(3,1,2)
plot(store(1,ind).time,store(1,ind).svpos)
xlabel('Time(sec)')
ylabel('Seeker Vert. Position(deg)')
grid

```

```

subplot(3,1,3)
plot(store(1,ind).time,store(1,ind).range)
xlabel('Time (sec)')
ylabel('Range (ft)')
grid
%=====
% Neural Network Missilie Dynamics from the Captive_Carry
Experiment

j=input('Enter the j value. "j" ranges from 1 to 8 --->
');
k= (j-1)*10;
ind=8+k;
ind1=ind-1;

pp=[store(1,ind).range';store(1,ind).shpos';store(1,ind).svp
os'];

len=round(0.97822*length(store(1,ind).mxpos));
p=[pp(:,1:1:len)];

factorp=max(max(p));
p(1,:)=p(1,:)/factorp;

a=sim(net,p);

figure(1)
subplot(2,1,1)
plot3(store(1,ind).mxpos,store(1,ind).mypos,store(1,ind).mzpo
s,'g',...
      a(1,:)*factorp,a(2,:)*factorp,a(3,:)*factorp,'r+',...
      store(1,ind-1).mxpos,store(1,ind-
1).mypos,store(1,ind-1).mzpos,'b-.',...
      store(1,ind-1).txpos,store(1,ind-
1).typos,store(1,ind-1).tzpos,'mo')
view(38,52)
orient tall
grid
xlabel('down range')
ylabel('cross range')
zlabel('altitude')
title([' Missile/Captive-Carry/Prediction Comparison -
Simulation 'num2str(ind)])

```

```

legend('Captive-Carry','Missile Dynamics Prediction','Actual
Missile')

subplot(2,2,3)
plot(store(1,ind).mxpos,store(1,ind).mypos,'g.',...
     a(1,:)*factorp,a(2,:)*factorp,'r+',...
     store(1,ind-1).mxpos,store(1,ind-1).mypos,'b-.',...
     store(1,ind-1).txpos,store(1,ind-1).typos,'mo')
orient tall
grid
xlabel('down range')
ylabel('cross range')

subplot(2,2,4)
plot(store(1,ind).mxpos,store(1,ind).mzpos,'g.',...
     a(1,:)*factorp,a(2,:)*factorp,'r+',...
     store(1,ind-1).mxpos,store(1,ind-1).mzpos,'b-.',...
     store(1,ind-1).txpos,store(1,ind-1).tzpos,'mo')
orient tall
grid
xlabel('down range')
ylabel('altitude')

figure(2)

subplot(3,1,1)
plot(store(1,ind).time,store(1,ind).shpos)
xlabel('Time(sec)')
ylabel('Seeker Hor. Position(deg)')
title([' Neural Network Inputs - Simulation ' num2str(ind)])
grid

subplot(3,1,2)
plot(store(1,ind).time,store(1,ind).svpos)
xlabel('Time(sec)')
ylabel('Seeker Vert. Position(deg)')
grid

subplot(3,1,3)
plot(store(1,ind).time,store(1,ind).range)
xlabel('Time(sec)')
ylabel('Range (ft)')
grid

real_miss_distance= store(1,ind-1).miss

```

```

xx1=(store(1,ind1).txpos);
xx2=(a(1,len)*factorp);
yy1=(store(1,ind1).typos);
yy2=(a(2,len)*factorp);
zz1=(store(1,ind1).tzpos);
zz2=(a(3,len)*factorp);
predicted_miss_distance=min(sqrt((xx1-xx2).^2+(yy1-
yy2).^2+(zz1-zz2).^2))-1000

figure(3)

plot3(a(1,:)*factorp,a(2,:)*factorp,a(3,:)*factorp,store(1,i
nd1).txpos, ...
      store(1,ind1).typos,store(1,ind1).tzpos)
view(25,65)
grid
xlabel('x')
ylabel('y')
zlabel('z')

```

LIST OF REFERENCES

1. Zarchan, P., *Tactical and Strategic Missile Guidance*, vol. 157, p. 26, American Institute of Aeronautics and Astronautics, Inc., 1994.
2. *Electronic Warfare and Radar Systems Engineering Handbook*, p. 10-1.2, Naval Air Systems Command and Naval Air Warfare Center Weapons Division, 1997.
3. *Electronic Warfare and Radar Systems Engineering Handbook*, p. 10-1.34, Naval Air Systems Command and Naval Air Warfare Center Weapons Division, 1997.
4. Rowe, A. W., *High Accuracy Distributed Sensor Time-Space-Position Information System for Captive-Carry Field Experiments*, p.31, Naval Postgraduate School, 1996.
5. *Electronic Warfare and Radar Systems Engineering Handbook*, p. 10-1.36, Naval Air Systems Command and Naval Air Warfare Center Weapons Division, 1997.
6. Barton, D. K., *Modern Radar System Analysis*, pp. 387-393, ARTECH HOUSE, Inc., 1988.
7. Brochure about Active Expendable Decoy System from AWA Defence Industries Pty Ltd.
8. Gill, C. W. and Pace, P. E., *Neural Prediction of Missile Dynamics During Hardware-In-The -Loop Captive-Carry Experiments*, Proceedings of the IEEE International Conference on Neural Networks, Houston, TX, 1997, pp. 2208- 2213.
9. Demuth, H. and Beale, M., *Neural Network Toolbox for Use with Matlab*, pp. 1-2, The MathWorks, Inc, 1998.
10. Hagan, M. T., Demuth, H. B., and Beale, M., *Neural Network Design*, Plus Publishing Co, MA, 1998.
11. Demuth, H., and Beale, M., *Neural Network Toolbox for Use with Matlab*, The MathWorks, Inc, 1998.
12. Hagan, M. T., Demuth, H. B., and Beale, M., *Neural Network Design*, pp. 12.19-12.23, Plus Publishing Co, MA, 1998.

13. Demuth, H. and Beale, M., *Neural Network Toolbox for Use with Matlab*, in Back Propagation, pp. 5-31, The MathWorks, Inc, 1998.
14. Gill, C. W., and Pace, P. E., *Neural Prediction of Missile Dynamics During Hardware-In-The-Loop Captive-Carry Experiments*, Proceedings of the IEEE International Conference on Neural Networks, Houston, TX, 1997, pp. 2209-2210.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218
2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5000
3. Superintendent 1
Attn: Professor Phillip E. Pace/Code EC/Pc
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5000
4. Superintendent 1
Attn: Professor Robert G. Hutchins/Code EC
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5000
5. Superintendent 1
Attn: Chairman, Electronic Warfare Academic Group/ Code IW
Naval Postgraduate School
Monterey, CA 93943-5000
6. Commanding Officer 1
Attn: Mr. Brian O'Connor/Code 5761.00
Naval Research Laboratory
4555, Overlook Avenue S.W.
Washington D.C. 20375-5339
7. Commanding Officer 1
Attn: Mr. Don W. Kahl/Code 5762.00
Naval Research Laboratory
4555, Overlook Avenue S.W.
Washington D.C. 20375-5339

8. Commanding Officer 1
Attn: Mr. Brian Edwards/Code 5760.00
Naval Research Laboratory
4555, Overlook Avenue S.W.
Washington D.C. 20375-5339
9. Commander 1
Attn: Mr. L. Peterson/Code PMA 272E1B
Naval Air Systems Command
Jacksonville, FL 32212-0122
10. Commander 1
Attn: Mr. Steve Wireman/Weapons Division Code/53C000D
Naval Air Warfare Center
China Lake, CA 93955
11. Commanding Officer 1
Attn: Mr. Anthony Link/Code 5765.00
Naval Research Laboratory
4555, Overlook Avenue S.W.
Washington D.C. 20375-5339
12. Commanding Officer 1
Attn: Mr. Bryan McGee/Code 5761.00
Naval Research Laboratory
4555, Overlook Avenue S.W.
Washington D.C. 20375-5339
13. LT Wagner A. de Lima Goncalves 1
589 E. Sampson Lane
Monterey, CA, 93940

REPRODUCTION QUALITY NOTICE

This document is the best quality available. The copy furnished to DTIC contained pages that may have the following quality problems:

- Pages smaller or larger than normal.**
- Pages with background color or light colored printing.**
- Pages with small type or poor printing; and or**
- Pages with continuous tone material or color photographs.**

Due to various output media available these conditions may or may not cause poor legibility in the microfiche or hardcopy output you receive.



If this block is checked, the copy furnished to DTIC contained pages with color printing, that when reproduced in Black and White, may change detail of the original copy.